Robust OCTs

Investigating Classification Tree Robustness

G. Lek

Bachelor Thesis

June 2022

TU Delft

Robust OCTs

Investigating

Classification Tree

Robustness

by

G. Lek

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on Tuesday June 22, 2022 at 15:30 PM.

Student number:	5073529	
Project duration:	April 19, 2022 – June 22 2022	
Thesis committee:	Dr. K.S. (Krzysztof) Postek	TU Delft, supervisor
	Dr. T.W.C. (Tom) Vroegrijk	TU Delft

TU Delft

Abstract

The application of machine learning in daily life requires interpretability and robustness. In this paper we try to make the process of building robust and interpretable decision trees more accessible. We do this by making the fitting of these models cheaper and simpler. We build on previous research and see if changing input data or the fitting formulation can create more robust trees that can be computed faster. To investigate this, we test whether data perturbations make heuristic algorithms more robust and whether enforcing constraints on adversarial examples in normal optimal classification tree MILP formulations can improve robustness. We also provide an altered formulation for the robust OCT model in Vos and Verwer (2021b) that yields better results with shorter runtimes. Finally, we extend the ROCT formulation to be applicable to multi-class classification and regression tasks.

Preface

This bachelor thesis report contains methods and results for fitting robust decision trees. The aim of this report is to obtain a Bachelor's degree in Applied Mathematics at Delft University of Technology. This project was supervised by Krzysztof Postek. I would like to thank Krzystof for weekly meetings and quick responses with thoughtful insights. Large parts of the code were borrowed from Daniël Vos' Github libraries [1]. Thanks to these libraries, I was able to quickly assess adversarial accuracy and create adversarial examples. Moreover, Daniël supported this project by accompanying us to 2 meetings and answering my emails in detail. I worked on this project with a lot of interest and enthusiasm because I was given the freedom to explore all my ideas without limits. Many of them turned out to be useless, but they helped me grasp the field of robust optimal decision trees. I hope the reader enjoys this bachelor thesis.

G. Lek
Delft, June 2020

[1] https://github.com/tudelft-cda-lab

Contents

1 Introduction

As research in artificial intelligence and machine learning advances rapidly, these models are increasingly being used in practice. Machine learning models are successful at prediction, but most lack interpretability and robustness (2.4).

Black box models are machine learning models that are not interpretable. When applying these in sensitive areas such as healthcare or cyber security, we lack insight into the decisions made. Decisions that impact people's lives need to be evaluated to ensure that these decisions are not made randomly or based on undesirable features. Interpretable models such as decision trees partially solve this problem. Interpretability in the context of a model has to do with being able to comprehend why certain decisions or predictions have been made.

Like most machine learning models, decision trees are susceptible to adversarial examples. This can lead to a malicious attacker exploiting these models. For example, if you predict someone's creditworthiness, that person could change a sensitive feature on the basis of which the prediction is made to borrow more money than is justified. The robustness of decision trees to these attacks brings us one step closer to the safe application of machine learning methods in our day to day operations.

Example 1.0.1. *Several billion-dollar NASDAQ-listed companies are applying artificial intelligence methods to assess credit risk and extend credit to consumers. Suppose these companies use a black box model. These models could place a high value on sensitive characteristics such as race. Without knowing whether this is the case, the American justice system cannot decide whether this model behaviour is acceptable.*

Now suppose we use a decision tree for credit risk assessment. We are now in a position to assess whether lenders have been treated fairly. But an adversary can exploit this tree by artificially inflating sensitive features like cash on hand by temporarily borrowing money from acquaintances.
A robust decision tree would be less susceptible to these attacks and reduce an adversary's ability to misuse the model.

1.1. Research objectives

In this Bachelor Thesis, we aim to investigate robustness of decision trees. Our main research question is:

”Can we obtain cheaper and simpler robustness of decision trees?”

Our report contains 3 main topics: simple perturbations for heuristic robustness, modifying an existing formulation for optimal robust trees to optimise faster and a simple algorithmic method to achieve robustness for a non-robust optimal tree formulation. In all of these issues, ways are being sought to achieve simpler and cheaper robustness. Even if some methods might prove unsuccessful, it is scientifically relevant to be able to exclude them. We formalize this in two sub-questions:

1. Can heuristic methods be made robust by simply changing the training data?
2. Can the formulation of ROCT (2.4.1) be changed or extended to make it more applicable and improve performance?

1.2. Contributions

Our first contribution is to evaluate whether we can obtain a robust decision tree by perturbing the training data of CART in section 4.1. Second, we present an algorithm for any OCT formulation that aims to make the tree more robust in section 4.1.5. Third, we propose an altered formulation for the ROCT formulation from Vos and Verwer (2021b) that optimises faster in section 5.1. Last, we propose novel formulations that extend ROCT: multi-class in section 5.1.3, regression in section 5.1.4.

1.3. Structure

In terms of structure, we first discuss the literature on robust optimal decision trees and then go into the data used and our implementation methods. Then we discuss the 2 sub-questions, addressing the methods used and the results obtained for each question. In the respective sections we try to answer our sub-questions. Finally, we conclude with a discussion and a conclusion where we answer our research question. When this paper talks about ”robust decision trees”, only the robustness against l_∞ attacks is considered.

1.3.1. Summary of notation used

OCT = Optimal Classification Tree

N = number of sample points

K = number of classes

p = number of features or reweighing scheme weight (clear from context)

X = arbitrary data set

L = number of subsets in partition

$X_i \in \mathrm{R}^p$ = sample i

y_i = class label sample i

$R_{xy}(T)$ = Misclassification error tree T

α = complexity parameter

T = arbitrary tree

N_{min} = minimum number of branch nodes

$N_x(l)$ = number of training points in leaf l $\in \mathcal{T}_B$

$\mathcal{T}_B$ = branch nodes

$\mathcal{T}_L$ = leaf nodes

CART = Classification and Regression Trees (heuristic algorithm)

GROOT = heuristic robust classification tree algorithm from Vos and Verwer (2021a)

ϵ = perturbation distance

Δ_l, Δ_r = left, right perturbation distances (equal to ϵ in l_∞ norm)

e_i = classification error sample i

$a_{jm} \in \{0, 1\}$ = node m splits on feature j $\in \{0, 1\}$

$b_m \in \mathrm{R}$ = threshold for node m

$s_{im0}, s_{im1} \in \{0, 1\}$ = sample i can travel left,right of node m respectively

$c_t \in \{0, 1\}$ = prediction for leaf t

$A_l(t), A_r(t)$ = left,right ancestor set of leaf t

$Z_{i,t} \in \{0, 1\}$ = node t reachable for sample i

d = maximum depth hyper parameter

X_{train} = training set

X_{test} = testing set

p_i = perturbation sample i

$N_{perturb}$ = number of perturbations

a_dist_i = adversarial distance

S_i = perturbation range sample i

C(x) = arbitrary classifier on arbitrary sample x

p(m) = parent node of node m

$l_i \in \mathrm{R}$ = absolute maximum loss

ROCT2 = altered ROCT formulation

2

Literature study

Classification tree learning is a modelling approach where the goal is to build a tree structure such that the branches represent divisions in the data and the leaves are class predictions for each data point. Machine learning research has developed powerful and fast algorithms for building classification trees. These trees are susceptible to adverse examples where tiny perturbations lead to misclassification of sample points. Learning classification trees is a NP-Hard (Hyafil and Rivest, 1976) problem. State-of-the-art methods generate optimally accurate decision trees for small data sets. Most of these methods use the framework of mixed-integer linear programs (MILP). Here, the target problem is formulated as a linear function with linear constraints that can be solved by MIP (Mixed integer program) solvers such as GUROBI 9.5.

2.1. The setup

Suppose we have N data points $(X_i, y_i), i = 1, ..., N, X_i \in R^n$. Where n is the number of features in the dataset. Our task is two-way classification on X_i, so we can generalize $y_i \in \{0, 1\}$. We want to partition the space R^n into L subsets, each defined as an intersection of half-spaces:

$$X_l = \{x \in \mathrm{R}^n : a_{l_k}^T \leq b_{l_k}, k \in 1, .., N_l\}$$

Defined for all $l \in 1, .., L$. We would like these subsets to satisfy two properties:

1. Mutually exclusive: ∀ l,l' ∈ 1,..,L it holds that $X_l \cap X_{l'}$ is empty or a subset of a hyperplane

2. Collectively exhaustive: We want X_l to partition the data set over all l :

$$\bigcup_l X_l = \mathrm{R}^n$$

To each subset X_l we assign a prediction label $y(X_l)$. We try to fit the partition and labels to the training data as well as possible.

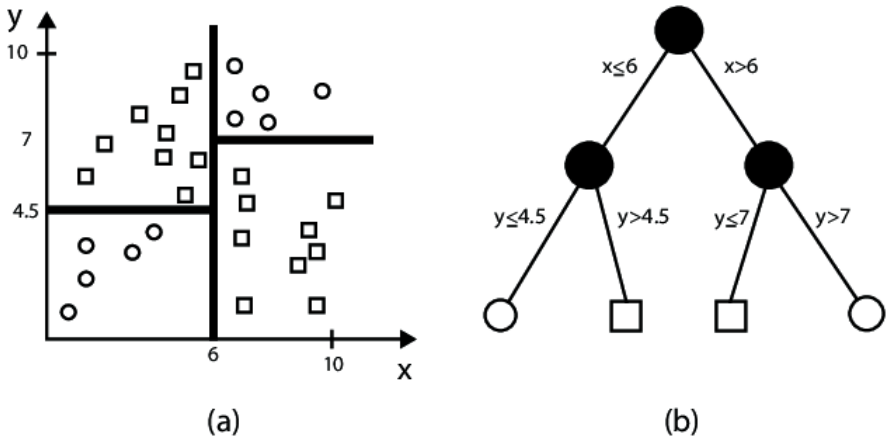


Figure 2.1: A binary classification task displayed as a partition and tree from Szücs and Schmidt (2018)

2.1.1. The general optimisation problem

Now assume that for every subset X_l we minimize a loss function $L(X_l, c_l)$ where c_l is a label generator. We arrive at the following optimisation problem:

$$\min_{a_{l_i}, b_{l_i}, c_l} \sum_{l=1}^{L} L(X_l, c_l) \tag{2.1}$$

$$s.t.\ X_l = \{x \in \mathrm{R}^n : a_{l_k}^T \leq b_{l_k}, k \in 1, .., N_l\} \tag{2.2}$$

$$X_l \cap X_{l'} = \emptyset\ or\ dim(X_l \cap X_{l'}) \leq n-1 \qquad \forall l \neq l' \tag{2.3}$$

$$\bigcup_l X_l = \mathrm{R}^n \tag{2.4}$$

Note that the feasible set is not convex. Moreover, it is incredibly difficult to formulate a closed-form expression for these constraints. Let us simplify this problem so that we can formulate it more easily. First organize the partitions by assuming that the number of subsets L is a power of 2, so that the subsets are formed by hierarchical partitions in a binary tree, see (b) in Figure 2.1. The root node (first node) splits the entire space with a single hyperplane, and the children of the root node split these halves. Applying this procedure d times, one obtains 2^d subsets of R^n.
Second, we restrict each partition to only one feature. This is exactly the way shown in (a) of 2.1. In this way, we restrict each partition to only one of n features, which drastically reduces the set of choices.

This interpretable tree is called a classification tree. Using the two aforementioned simplifications, we can model the problem as a mixed-integer linear optimisation problem (MILP) . This formulation was first presented in Bertsimas (2017) and is discussed in section 2.3. Solving the MILP problem is computationally expensive, which means that we cannot yet apply this method to large data sets in a reasonable amount of time.

2.2. Heuristics

Before Bertsimas & Dunn in Bertsimas (2017) and S. Verwer in Verwer and Zhang (2017), attempts to fit classification trees did not provide optimality, but used heuristics to solve the problem. Instead of optimizing the entire tree in one step, these algorithms recursively created locally optimal hyperplanes.

As a first step we think about a partition with one hyperplane and then create a hyperplane in the subsets created by this split. For this hyperplane, we use a loss function that tries to partition the

subsets ”as purely as possible”. The splitting of the nodes in this algorithm is parallelizable, making the algorithm fast and scalable.

The algorithm CART (Breiman et al., 1983) uses a loss function with Gini impurity as a metric. This metric is a measure of how often a sample is mislabeled when selected by the distribution of labels in the subset. The Gini impurity can be calculated by summing the probability p_i that a sample with label i is selected and multiplying by the probability $1 - p_i$ that the sample is mislabeled. For classification tasks with two classes, the Gini impurity is :

$$L_G(p) = p_0(1 - p_0)p_1(1 - p_1) = 1 - (p_0 + p_1)$$

2.3. MILP Formulation

The MILP formulation of Bertsimas (2017) produces an optimal classification tree when the optimisation is complete. This is formalised as a MILP problem and can be solved with any MIP solver. Instead of local optimisation at each partition, this method optimises the entire tree at once, allowing optimality to be achieved. Instead of an impurity measure used in inductive top-down methods, the formulation uses a more natural misclassification objective. The whole tree is formed in one step, i.e. the splits are formed with the knowledge of the other splits in the tree. The MILP formulation tries to solve the problem:

$$\min_{x,y} \quad R_{xy}(T) + \alpha|T| \tag{2.5}$$

$$\texttt{s.t.} \quad N_x(l) \geq N_{min} \tag{2.6}$$

Where $R_{xy}(T)$ is the misclassification error of tree T, |T| is the number of branch nodes. $N_x(l)$ is the number of training points in leaf l, with a minimum of N_{min}. α is a complexity parameter that controls the depth of the tree. For the full formulation and explanation of each constraint, I refer the reader to Bertsimas (2017), pages 1039-1082. The main result of this model is that it achieves optimality on the training set but is inefficient. This means that this model cannot be applied to large data sets.

2.3.1. Novel formulations since 2017

Many efforts have been made to make the OCT (Optimal Classification Tree) formulation more efficient and accurate. In this section we will briefly discuss the contributions of these efforts.

Firat et al. (2020) has created an ILP formulation that applies a column generation heuristic to scale the problem. Similarly, Verwer and Zhang (2019) has addressed the dependence on the size of the data set by formulating the variables using binary encoding. Both methods scale to larger data sets and provide competitive accuracy.

Blanquero et al. (2021) created a randomised continuous optimisation formulation with probabilities instead of deterministic decisions at the branching nodes. This converges to the result of Bertsimas (2017) and scales well with the size of the training set.

Aghaei et al. (2021) has developed an extremely fast max-flow based formulation using Benders decomposition. This method essentially splits the problem into a main problem and a sub-problem, with the sub-problem being fast to solve. This formulation has a robust variant presented in Justin et al. (2022). These formulations scale extremely well and are much faster than the original OCT MILP formulation. The disadvantage of this formulation is that it can only handle binary features. A data set with only binary features is rare in practical applications. This can be solved by binning continuous/integer variables, but you lose information or get extremely large and non-interpretable trees.

A robust version of inductive local optimal algorithms like CART was developed in Chen et al. (2019) and further improved in Vos and Verwer (2021a). This algorithm ”GROOT” is fast and yields competitive robust results.

2.4. Robustness

Artificial Intelligence (AI) robustness deals with the study of defence against adversarial examples, these are samples created with the goal of misclassifying the model.

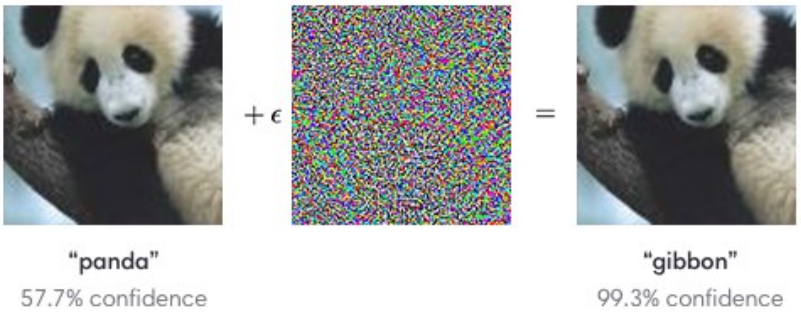


Figure 2.2: Example of adversarial attack from Goodfellow (2020)

In Figure 2.2 we see the example of an adversarial example of an image classifier created by some perturbation ϵ. After perturbing, the image is misclassified as ”gibbon” but should be classified as ”panda”. The setting of robust learning presents us with a min-max optimisation problem:

$$\min_{\theta} \mathrm{E}_{(x,y)\sim D}\left(\max_{\delta\in S} L\left(\theta, x+\delta, y\right)\right) \tag{2.7}$$

We want to find parameters θ that minimizes loss function L over feature variable x and class variable y from distribution D. The attacker maximizes the loss by changing samples $(x, y) \sim D$ with perturbations $\delta \in S$. S is predefined and contains the allowed perturbations. In the work of Vos and Verwer (2021b) it is proven that equation (2.7) under l_∞-attacks can be reformulated as a single minimisation over θ:

$$\min_{\theta} \sum_{(x,y)\sim D}\left[\bigvee_{t\in\mathcal{T}_L^{S(x)}} c_t \neq y_t\right] \tag{2.8}$$

Where c_t is the prediction label of leaf t and $\mathcal{T}_L^{(S(x)}$ is the set of all leaves reachable by perturbations on x.

2.4.1. ROCT

ROCT is a novel robust version of the original OCT formulation, introduced by Vos and Verwer (2021b). ROCT uses l_∞ (supremum norm) perturbations to find the optimal tree that is robust to the given perturbation range $[-\epsilon, \epsilon]$.

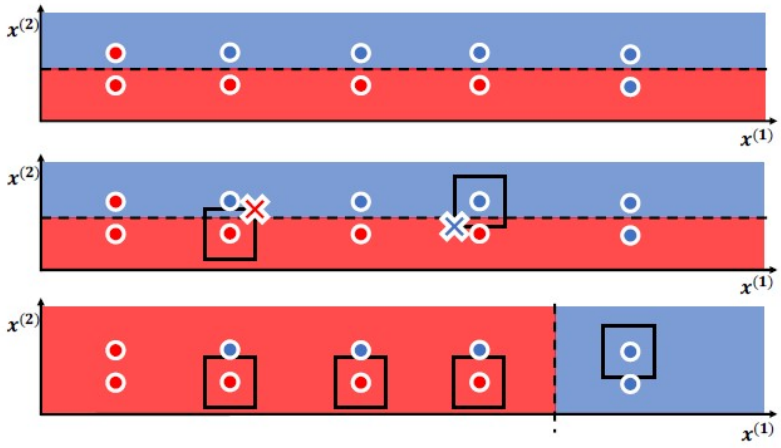


Figure 2.3: Illustration of robust splitting with l_∞ ball around 10 samples from Chen et al. (2019)

In Figure 2.3 we see how an optimal split is defined for such a data set. The boxes around each sample is the l_∞ ball of radius ϵ. The crosses are examples of places where an adversarial example can be generated/exists within this ball. If all samples are perturbed adversarially in Figure 2.3, the accuracy would be 0. The lower illustration in Figure 2.3 splits on $x^{(1)}$ which yields a robust split, with worst case accuracy after adversarial perturbations of 0.7.

A more extensive explanation on notation and constraints can be found in Vos and Verwer (2021b). Our objective is to train a robust tree of depth d, our objective function inspired by equation (2.8) is to minimize misclassifications over all samples: $\sum_{i=1}^{n} e_i$. We restrict our tree to selecting on one feature per branch node: $\sum_{j=1}^{p} a_{jm} = 1$ where a_{jm} is a binary variable deciding on what feature j branch node m splits. Our threshold values are stored in continuous variables b_m for branch nodes m. Together with perturbation radii, we can create the condition for s_{im0} and s_{im1}. These are binary variables deciding whether sample i can move left and right of node m in the set of branch nodes $\mathcal{T}_B$ respectively:

$$\left(\vec{X_i} - \vec{\Delta^l}\right) * \vec{a_m} \leq b_m \implies s_{im0} \quad \forall m \in \mathcal{T}_B, i = 1,..N \tag{2.9}$$

$$\left(\vec{X_i} + \vec{\Delta^r}\right) * \vec{a_m} > b_m \implies s_{im1} \quad \forall m \in \mathcal{T}_B, i = 1,..N \tag{2.10}$$

Where $\vec{\Delta^l}, \vec{\Delta^r}$ are vectors of the left,right perturbations, $\vec{X_i}$ is the row vector of feature values from sample i and $\vec{a_m}$ is a vector of all 0's except on the index of the splitting feature. The left hand side of the above inequalities represents the value of the branch feature for the perturbed sample. For simplicity, we will not consider vector notation.

To determine the misclassifications, we create a constraint which forces $e_i = 1$ if for any leaf t, sample i can reach t and the sample is misclassified:

$$\bigwedge_{m \in A_l(t)} s_{im0} \bigwedge_{m \in A_r(t)} s_{im1} \wedge [c_t \neq y_i] \implies e_i \quad \forall m \in \mathcal{T}_L, i = 1,...N \tag{2.11}$$

Where $A_l(t), A_r(t)$ are the left and right ancestors of leaf t respectively.
These constraints form the full ROCT MILP formulation. The first term in the left hand side of the implication forces reach-ability of sample i to leaf t. The formulation is dominated by the s variables with $2^d n$ variables. There is also a binary formulation with no continuous thresholds, found in Vos and Verwer (2021b).

3

Data & implementation

In this section we discuss the data used and how it is implemented.

3.1. Programming ROCT & CART

The implementation of ROCT and CART has been done using Python 3.8. The programming consists of three parts:

1. Data import and transformations
2. MILP implementation and solving using optimizer or running fitting algorithm
3. Tree modelling and assessment

3.1.1. Data

We used eight data sets from OpenML (Feurer et al., 2019), which were also used in Vos and Verwer (2021b), see table 3.1. All training sets are first scaled using Min-Max scaling for each feature. This is done as follows: $x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$, this scales the features to a $[0, 1]$ range, which means our l_∞ perturbation ϵ can also be restricted to $[0, 1]$. We split our data sets into training, testing sets with a split of 80%,20% . Our testing set: X_{test} is scaled with the Min-Max transformation parameters of the training set.

Table 3.1: Overview of samples,features and majority class ratio across all used data sets in our experiments

Name	Samples	Features	Ratio majority class
Haberman	306	3	.735
Blood-transfusion	748	4	.762
Cylinder-bands	277	37	.643
Diabetes	768	8	.651
Ionosphere	351	34	.641
Banknote-authentication	1372	4	.555
Breast-cancer	683	9	.650
Wine-quality	6497	11	.633

For each data set, three epsilons(perturbation radii) and one depth parameter are computed as in Vos and Verwer (2021b). The depth parameter is computed using 3-Fold Stratified Cross-Validation: We split the data set into three parts and then use 2/3 as the training set and the other 1/3 as the test set. We do this for all combinations of epsilon with d's and choose the parameter with the best performance across all sets.

The ϵ is chosen by calculated upper bounds of adversarial accuracy. This method is first introduced in Vos and Verwer (2021b). The upper bound is determined by creating a graph G = (V,E), with V =

data set X and an edge $e \in E$ when the perturbation regions of samples overlap. Maximum matching then yields an upper bound on the adversarial accuracy. Epsilons are chosen at 20%, 50%, 75% of the range of bounds when ϵ ranges from 0 to 1.

3.1.2. Optimisation

All optimisations are carried out with MIP solver GUROBI. The running time is limited to 30 minutes. After implementing the model myself, Daniël Vos' Github library is used for the implementation. This is because this implementation conveniently converts the GUROBI output into an evaluable tree data structure. For implementation, the formulation presented in section 2.4.1 has to be converted to MILP form.

(2.10) is translated to MILP by using big M constraints for the implications and the strict inequality is adjusted using a sufficiently small number ψ:

$$\left(\vec{X_i} - \vec{\Delta^l}\right) * \vec{a_m} \geq b_m + \psi - M s_{im0} \tag{3.1}$$

$$\left(\vec{X_i} + \vec{\Delta^r}\right) * \vec{a_m} \leq b_m + M s_{im1} \tag{3.2}$$

Also equation (2.11) is turned into MILP form by first creating an if statement for y_i to be 0 or 1. If $y_i = 0$ then we force e_i to be 1 correctly by:

$$e_i \geq \sum_{m \in A_l(t)} s_{im0} \sum_{m \in A_r(t)} s_{im1} + c_t - 1 \tag{3.3}$$

similarly for $y_i = 1$ we use the same formula but with $(1 - c_t)$ instead of c_t.

CART implementation

We used the CART implementation of Scikit-Learn (Pedregosa et al., 2011) and applied 3-Fold Stratified Cross-Validation for the depth hyper parameter in the perturbation schemes. We then translated the fitted tree into GROOTs' tree data structure for evaluation.

3.1.3. Tree modelling and assessment

We use the tree structure from Daniël Vos' GROOT package and test this tree for accuracy on the test set. Our measure for evaluating adversarial robustness to l_∞ attacks is the adversarial accuracy score. This score is calculated by determining for each pair of samples whether their regions intersect. The formula is:

$$\frac{(1 - \#\{i \in X_{test} | \text{i s.t. a feasible attack is possible}\})}{\#X_{test}} \tag{3.4}$$

4

Simple robust heuristic methods

In this section we try to answer the question:

”Can heuristic methods be made robust by simply changing the training data?”

We do this by proposing several schemes to try to make CART and OCT formulations more robust. This is done through changing the training data, and also by trying an algorithm to make any OCT formulation more robust. Afterwards, we compare our results with GROOT and ROCT in terms of the accuracy and adversarial accuracy (see formula (3.4)) metrics on our test data. We also vary the number of perturbations from 0 to $N_{perturb}$ to see the effect of additional perturbations.

4.1. Methods

Formulations like ROCT in Vos and Verwer (2021b) and GROOT in Vos and Verwer (2021a) improve robustness by implementing it in the model itself. Our main area of research in this section is whether this robustness can also be achieved with a simpler idea: perturbing data and fitting the fast CART algorithm to that data, and enforcing misclassification constraints on adversarial examples in an OCT formulation. We propose four different CART perturbation schemes and analyse the best performing one of them. We also present an algorithm to attain robustness of OCT formulations.

4.1.1. Uniform perturbations

The most straightforward method would be to sample from a uniform distribution on $[-\epsilon, \epsilon]$ and perturbing our data with this. In our method we sample $p_i \sim U_{[-\epsilon,\epsilon]}$ and for each sample X_i we perturb it with $X_i + p_i$ to get a perturbed sample which we add to our training data set X_{train}. We do this $N_{perturb}$ times to get a training data set of size $(N_{perturb} + 1) * \#\{X_{train}\}$.

4.1.2. Linear perturbations

To remove the uncertainty in the results, we use a second method that generates perturbations in a range of $[X_i - \epsilon, X_i + \epsilon]$. For the first perturbation, these are simply the points $X_i - \epsilon, X_i, X_i + \epsilon$. The jth perturbation divides the range into $j \in 1, \ldots, N_{perturb}$ points to which X_i is added. For the jth perturbation, we obtain the perturbed samples: $X_i - \epsilon, X_i - \epsilon + \frac{2\epsilon}{k}, X_i + \epsilon$ for $k \in 1, \ldots, j-2$. This is simply an equally distributed set of perturbed samples with perturbations in $[-\epsilon, \epsilon]$. Again, we add the perturbed samples to X_{train} and do this for $j \in 1, \ldots, N_{perturb}$.

4.1.3. Adversarial examples

For our next scheme, we generate an adversarial example for each perturbation and create a new training set X_{adv} to which we fit CART. X_{adv} is a concatenation of X_{train} and its adversarial examples. We repeat this $N_{perturb}$ times. In this way, we ”walk” through the successive adversarial examples in each iteration. This method however, leads to unstable results, as the path of each sample point can move away from the ideal robust split of the original X_{train}.

4.1.4. Reweighing scheme

For our next data alteration, we apply a reweighing scheme to X_{train} in an attempt to make the samples that can generate adversarial examples more important in fitting. We calculate the minimum perturbation distance for each point to become an adversarial example: a_dist_i. We multiply each sample X_i by $(1 - a\_dist_i)^p$, experimenting with increasing powers of p. If the sample X_i is an adversarial example, it remains unchanged since $a\_dist_i = 0$ and its weight is therefore 1^p. For other samples, points with smaller adversarial distance are weighted than points with larger distance. For each p, we perform $N_{perturb}$ iterations in which we multiply and refit X_{train} by its reweighing scheme $(1-a\_dist_i)^p$. We let ROCT run for 30 minutes after which we compute the accuracy and adversarial accuracy scores.

4.1.5. Restart OCT with adversarial examples

Instead of forcing robustness within the formulation, as is the case with ROCT, we can try to enforce robustness in a more indirect way. Note that it is quite difficult to compute adversarial examples, and our method does this heuristically. Optimal adversarial examples have been computed in Kantchelian et al. (2015), but this can take quite a long time. We fit an OCT and try to implement robustness in a clever way. The algorithm looks as follows when applied to an OCT MILP:

1. While critical point not reached w times
2. Start optimisation of OCT MILP
3. Stop if incumbent solution is found
   (a) Generate adversarial examples on X if feasible within perturbation range
   (b) Add adversarial examples to X_{train}
   (c) Force adversarial example (X_{i^*}) of sample $(X_i \in X)$ to have equal misclassification with constraint: $e_{i^*} = e_i$
4. Repeat from step 1 with new constraints and sample points from 3c

In this algorithm, we initialise with the number of critical points to be reached under the variable w. We define a critical point for the restart OCT as the iteration where the objective function and the accuracy score have not increased. Currently, we have no means to obtain an proof of optimality. We can evaluate the trade-off between accuracy and adversarial accuracy in a graph where the scores for each iteration are plotted.

4.2. Results

In this section we discuss and compare the results of our best performing scheme and inspect our reweighing scheme. The results for the other mentioned schemes applied to CART can be found in the Appendix. We also analyse the results of the restart OCT at each iteration.

4.2.1. Linear perturbations

We let ROCT run for 30 minutes after which we compute the accuracy and adversarial accuracy scores. Full tables with scores for every data set and epsilon are attached in the Appendix. Our linearly perturbed CART scheme is referred to here as ”P-CART” for convenience.

Table 4.1: Linearly perturbed CART mean score comparison with $N_{perturb} = 10$ for P-CART

Score	ROCT (1800s)	GROOT	P-CART
Mean adv. accuracy	0.724	0.717	0.596
Mean accuracy	0.754	0,767	0.658

In Table 4.1 we see the mean scores across all data sets and perturbation radii ϵ. When we run P-CART for 10 iterations, we have a mean adversarial accuracy increase against CART of 0.149. Against ROCT, we have 3/24 wins and one tie regarding adversarial accuracy. Against GROOT we have

3/24 wins and three ties. Computation time for ROCT is 1800s whereas GROOT and P-CART compute within seconds. We notice that for data sets with larger depth hyper parameters, our method performs better.

Considering only data sets and epsilon combinations with depth ≥ 3, we receive an mean adversarial accuracy increase of 0.225 on these data sets. The mean difference between ROCT adversarial scores and ours is 0.035, with 3/6 wins and one tie. Then, regarding accuracy we notice three wins and two ties, where we have 0.079 higher mean accuracy . The results suggest that our method does not work well with nearly trivial trees, since there are a lot of depth 0 and 1 trees where P-CART scores poor. ROCT can quickly reach optimality at these depths, as the formulation grows exponentially with depth. For visualization purposes: considering only trees of depth ≥ 3, we plot the adversarial accuracy score against the amount of perturbations.

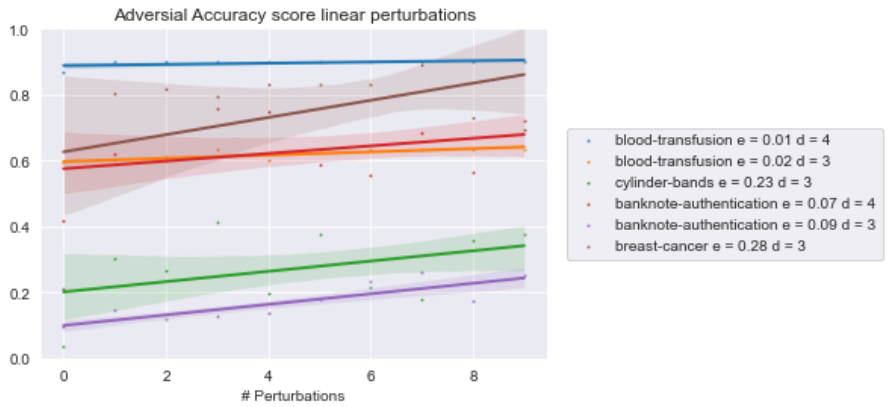


Figure 4.1: Data set, epsilon combinations with depth ≥ 3 for the linear perturbation scheme with CART. Linear regression line with 95 % confidence interval added in plot.

In Figure 4.1 we can see the increase in adversarial accuracy with increasing perturbations. Further perturbations have diminishing effects: the mean increase in adversarial accuracy from perturbation 10 to 100 is 0.00075.

4.2.2. Reweighing scheme

Following our intuition, the reweighing scheme from section 4.1.4 should positively influence our adversarial accuracy score. Reweighing for 20 iterations for $p \in 0, 1, ..., 50$ and plot the power p against the mean increase in adversarial accuracy score.

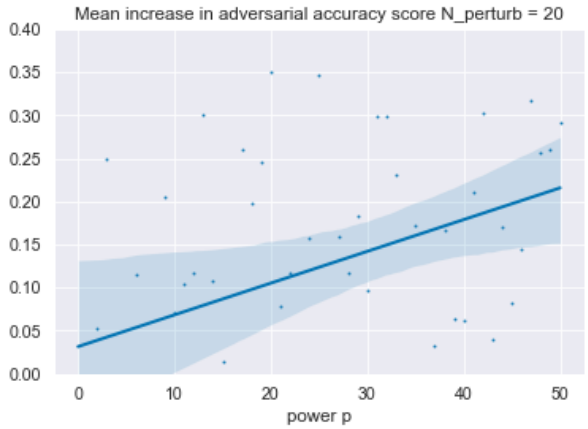


Figure 4.2: Regression plot with 95 % confidence interval of power p in $(1 - a\_dist_i)^p$ against the mean increase in adversarial accuracy across 20 iterations

In Figure 4.2 we see a clear positive relationship, but we also find unstable results, which can be seen in the large confidence interval and spread of points. The randomness of CART in combination with ”walking” over successive adversarial distances may cause this instability.

4.2.3. Restart OCT

For the restart OCT algorithm, we record the adversarial accuracy and the objective function at each iteration. We choose $w = 2$ so that the algorithm stops after 2 critical points. This means that we do not find an improvement in the objective function or the adversarial accuracy two times. The OCT MILP model itself optimises accuracy only via the objective function, and the adversarial accuracy is only captured by the additional constraints in each iteration. For this reason, we find that the algorithm constantly trades accuracy for adversarial accuracy and vice versa, as seen in Figure 4.3 below:

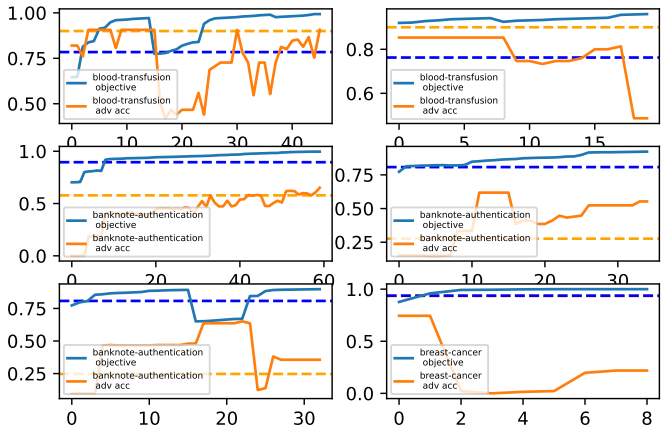


Figure 4.3: Data set, epsilon combinations with depth ≥ 3 with adversarial accuracy and the objective function value plotted at each iteration. The blue, orange dashed line represents the ROCT score for training accuracy, adversarial accuracy respectively. In the breast- cancer graph, both dashed lines are at the same level.

In Figure 4.3 we see that the algorithm often sacrifices adversarial accuracy for accuracy (objective

value) score. In practical applications, we can stop the algorithm at a desirable point where the accuracy and the adversarial accuracy are sufficiently high. We find that the accuracy for each graph in the termination iteration is close to 100%. In most cases, higher adversarial accuracy is desirable instead of optimal training accuracy.

If we compare the ROCT (dashed line) scores with the plot of the restart OCT, we find that stopping the algorithm at certain points can yield better results. Applied to practice, we stop the algorithm before it over-optimises for accuracy and loses adversarial accuracy.

4.3. Data alterations for heuristic robustness

In this section we try to answer our research sub-question:

”Can heuristic methods be made robust by simply changing the training data?”

We find that heuristic methods without robustness in their formulations or loss functions can be made more robust by changing their training data. However, models with robustness in their formulation achieve better results and robust heuristics (GROOT) outperform our heuristic perturbation schemes. We can balance the adversarial accuracy with our proposed restart OCT to obtain a more robust model with any OCT formulation. This algorithm is simple to apply to any OCT model.

5

Altering the ROCT formulation

In this section we attempt to answer the question:

”Can the formulation of ROCT be changed or expanded to make it more applicable and improve performance?”

We do this by first altering the ROCT formula so that it computes faster. Second, we extend ROCT so that it is applicable for regression and multi-class classification.

5.1. Methods

5.1.1. Altering ROCT formulation

Consider the ROCT formulation from section 2.4.1. We will try to strengthen this formulation by adding valid inequalities that exploit symmetry and valid bounds.
Given a classifier $C(x) \longrightarrow \{0, 1\}$ note that a sample i can only be correctly predicted against an adversary if its entire perturbation range S_i is correctly predicted.

$$\forall x \in S_i : C(x) = y_i \tag{5.1}$$

In Vos and Verwer (2021b) a valid inequality is introduced, derived from the notion that two samples from different classes with intersecting perturbation ranges cant both be correctly classified. Thus for sample i,j $\in 1, .., N$ s.t. $i \neq j$:

$$y_i \neq y_j \text{ and } S_i \cap S_j \neq 0 \implies e_i + e_j \geq 1 \tag{5.2}$$

As our second alteration, we will leverage the binary tree structure. We observe that if a sample is unable to move left from the root node, the sample cant move to any leaf nodes in the left sub tree. Thus we can force the leaf t to be unreachable:

$$s_{i,1,0} = 0 \implies \sum_{t \in t_1, \ldots, t_{\frac{|\mathcal{T}_L|}{2}}} Z_{i,t} = 0 \quad \forall i \in 1, \ldots, N \tag{5.3}$$

$$s_{i,1,1} = 0 \implies \sum_{t \in t_{\frac{|\mathcal{T}_L|}{2}}, \ldots, t_{|\mathcal{T}_L|}} Z_{i,t} = 0 \quad \forall i \in 1, \ldots, N \tag{5.4}$$

Where $Z_{i,t} = 1$ if leaf t is reachable for sample i. Instead of including these conditions in our formulation, we implement them by changing the definition of $Z_{i,t} \in \{0, 1\}$. In the below constraint, $Z_{i,t}$ for leaf t in the left sub-tree is 1 if it is reachable and $s_{i,1,0}$ is 1. This way we don’t increase the number of constraints. We use $Z_{i,t}$ instead of the ”and” wedges on the left-hand side of equation (2.11). The

constraint is formalised as:

$$\sum_{m\in A_l(t)} s_{i,m,0} \sum_{m\in A_r(t)} s_{i,m,1} - \#\{A_l(t) \cup A_r(t)\} + s_{i,1,0} \leq Z_{i,t} \quad \forall i \in 1,\dots,N \; \forall t_1,\dots,t_{\frac{|\mathcal{T}_L|}{2}} \tag{5.5}$$

$$\sum_{m\in A_l(t)} s_{i,m,0} \sum_{m\in A_r(t)} s_{i,m,1} - \#\{A_l(t) \cup A_r(t)\} + s_{i,1,1} \leq Z_{i,t} \quad \forall i \in 1,\dots,N \; \forall t_{\frac{|\mathcal{T}_L|}{2}},\dots,t_{|\mathcal{T}_L|} \tag{5.6}$$

When the s-variables for the root node are 0, then (5.5) and (5.6) are rendered useless, since $Z_{i,t} \in \{0,1\}$ is always ≥ 0.

The third modification takes advantage of the structure of s-variables. We note that for branch nodes with branch nodes as children, a sample that cannot reach the parent branch node certainly cannot reach the children branch nodes. This gives us the idea for inequalities: $s_{i,m,0} \geq s_{i,c_l(m),0}$, $s_{i,m,0} \geq s_{i,c_l(m),1}$. This also holds for $s_{i,m,0}$ and its right child. To incorporate this relation, we replace equations (2.9),(2.10) for all branch nodes except the root with the following implication, modelled with big M constraints:

$$\begin{cases} s_{i,p(m),0} = 1 \implies \text{(2.9), (2.10) if m left child of p(m)} \\ s_{i,p(m),1} = 1 \implies \text{(2.9), (2.10) if m right child of p(m)} \end{cases} \tag{5.7}$$

Where p(m) is the parent node of branch node m. Again, we do not add constraints but just alter existing constraints. Adding the proposed constraints made our model slower due to a larger formulation.

5.1.2. Unused Alterations

We thought of several valid inequalities that did not improve the empirical optimisation time. These are listed here with a brief explanation.

First, we considered an inequality that should always hold for leaves. Both leaves should not predict the same class if they have the same parent branch node. If both leaves predict the same class, their parent branch node is superfluous, as there should always be a split that improves fit. If this is not the case, the split can, in extreme cases, send all samples to one leaf and leave the other empty. This is formalised as follows:

$$c_{t_0} + c_{t_1} = 1 \quad \forall t_0, t_1 \in \mathcal{T}_L \text{ s.t. } p(t_0) = p(t_1) \tag{5.8}$$

Second, we attempt to force the inequality on which equation (5.7) is based, to directly bound all s variables:

$$s_{i,C_l m,0} \leq s_{i,m,0} \text{ , } s_{i,C_l m,1} \leq s_{i,m,0} \quad \forall m \in \mathcal{T}_B \text{ s.t. } C_l(m), C_r(m) \in \mathcal{T}_B \tag{5.9}$$

$$s_{i,C_r m,1} \leq s_{i,m,1} \text{ , } s_{i,C_r m,1} \leq s_{i,m,1} \quad \forall m \in \mathcal{T}_B \text{ s.t. } C_l(m), C_r(m) \in \mathcal{T}_B \tag{5.10}$$

These constraints did not improve performance, possibly due to the increasing size of the formulation.

Third, we have implemented the constraints in equation (5.3) directly by including them in the formulation, and we have extended them for branch nodes other than the root. We give an example of these constraints for the left branch node of the second depth:

$$s_{i,2,0} = 0 \implies \sum_{t\in t_1,\dots,t_{\frac{|\mathcal{T}_L|}{4}}} Z_{i,t} = 0 \quad \forall i \in 1,\dots,N \tag{5.11}$$

$$s_{i,2,1} = 0 \implies \sum_{t\in t_{\frac{|\mathcal{T}_L|}{4}},\dots,t_{\frac{|\mathcal{T}_L|}{2}}} Z_{i,t} = 0 \quad \forall i \in 1,\dots,N \tag{5.12}$$

This is repeated for each branch node with children branch nodes.

5.1.3. multi-class ROCT

In this subsection we briefly introduce a novel multi-class version of ROCT (Vos and Verwer, 2021b). We obtain a multi-class ROCT formulation by changing constraint (2.10). The original constraint in MILP form is:

$$e_i \geq \begin{cases} \sum_{m \in A_l(t)} s_{im0} \sum_{m \in A_r(t)} s_{im1} + c_t - 1 \text{ , if } y_i = 0 \\ \sum_{m \in A_l(t)} s_{im0} \sum_{m \in A_r(t)} s_{im1} + (1 - c_t) - 1 \text{ , if } y_i = 1 \end{cases} \quad \forall t \in \mathcal{T}_L,\ i \in 1, \ldots N \tag{5.13}$$

Instead of a binary leaf predictor variable $c_{k,t}$ we use a K-dimensional binary vector, with K the number of classes: $c_{k,t} \in \{0,1\}^K$ for each leaf t Where $c_{k,t} = 1$ if leaf t predicts class k. Next we formulate the constraints:

$$e_i \geq Z_{i,t} + (1 - c_{k,t}) - 1 \text{ , if } y_i = k \quad \forall t \in \mathcal{T}_L,\ i \in 1, \ldots N,\ k \in 1, .., K \tag{5.14}$$

$$\sum_{k \in 1, \ldots, K} c_{k,t} = 1 \quad \forall t \in \mathcal{T}_L \tag{5.15}$$

If $c_{k,t}$ predicts $k \in 1, \ldots, K$, this binary variable $c_{k,t}$ and only this one equals 1 for the respective leaf. In the case that leaf t is reachable ($Z_{i,t} = 1$), but $c_{k,t} = 0$ for the class of sample i, then $e_i \geq 1 + 0$. This constitutes a misclassification. In all other cases $e_i \geq 0$, so e_i is minimised in the objective to 0. We relax e_i to be a continuous variable.

5.1.4. Regression ROCT

In this section we create a robust optimal regression tree algorithm from ROCT. The goal is to predict continuous values in each leaf, since the class labels in the regression setting are continuous. To do this, we use maximum absolute loss, i.e. we minimise the maximum absolute loss across leaves. We create variables:

1. $l_i = \max_{t \in \mathcal{T}_L} \{|e_{i,t} - c_{i,t}|\}\ s.t.\ i \in 1, \ldots, N$,
2. $e_{i,t} = c_t * Z_{i,t}\ s.t.\ i \in 1, \ldots, N\ t \in \mathcal{T}_L$,
3. $c_t \in \mathrm{R}\ s.t.\ t \in \mathcal{T}_L$ (continuous prediction in leaf t)

We linearise with auxiliary variables and inequalities instead of the absolute value.

$$L_{i,t}^{auxiliary} = |e_{i,t} - c_{i,t}| \iff L_{i,t}^{auxiliary} \geq e_{i,t} - c_{i,t} \text{ and } L_{i,t}^{auxiliary} \geq -e_{i,t} + c_{i,t} \quad \forall i \in 1, \ldots, N \tag{5.16}$$

$$L_i = \max_{t \in \mathcal{T}_L} \{L_{i,t}^{auxiliary}\} \quad \forall i \in 1, \ldots, N \tag{5.17}$$

The maximum is linearised with big M constraints. Our objective function is the minimisation of $\sum_{i=1}^{N} L_i$. In this way we minimise the sum of the maximum absolute losses.

5.2. Results

5.2.1. Altered ROCT formulation

We altered the formulation with the goal of achieving better performance with shorter runtime. To assess this we run both the original model together with the altered one ("ROCT2") on different time limits and assess their scores. In Figure 5.1 we see both models run on 1000 seconds with a logarithmic x axis and the mean training error on the y axis.

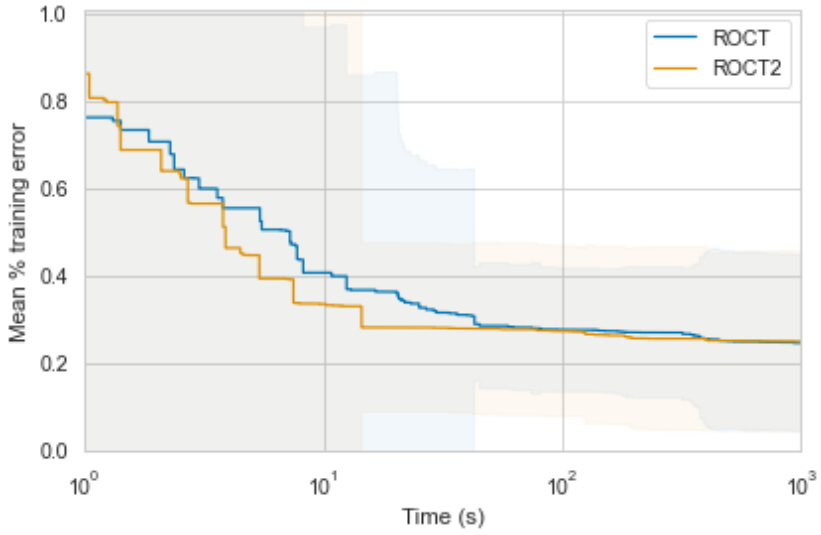


Figure 5.1: Mean percentage misclassified training samples of ROCT and the altered ROCT (ROCT2) formulation with all data set, epsilon combinations s.t depth ≥ 2. Where colored ranges represent one standard error.

Note in Figure 5.1 that ROCT2 optimises faster on average, but after about 100 seconds both models follow roughly the same path. This plot only takes into account the ≥ 2 depths, otherwise most of the data sets quickly fall towards a very small training error since the tree has ≤ 1 splits.

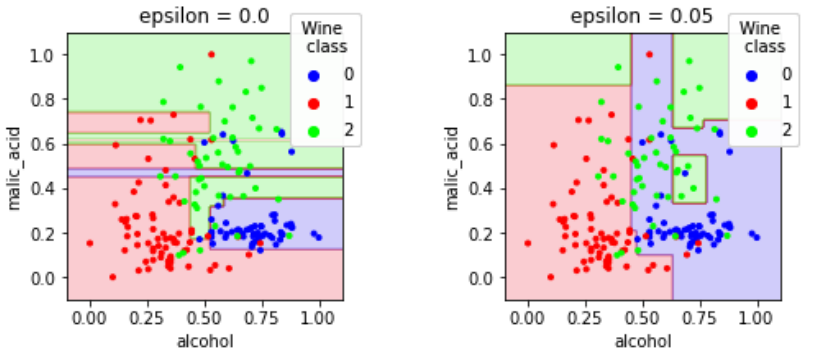
Table 5.1: mean (adv. accuracy, accuracy) scores for four different runtimes of the altered ROCT formulation. Wins are in bold.

Runtime	ROCT	ROCT2
60s	(0.612,0.619)	(**0.671**,**0.682**)
180s	(0.700,0.726)	(**0.702**,**0.731**)
300s	(0.700,0.728)	(**0.724**,**0.761**)
1800s	(0.724,0.754)	(**0.726**,**0.763**)

In Table 5.1 we see that the altered formulation wins over the normal formulation in all cases, but at 1800s both means are almost equal.

5.2.2. multi-class ROCT

For the multi-class ROCT, we plot the splits fitted to the wine data set, a data set where there are three wine types as classes. We will visually compare the normal multi-class class OCT and the multi-class ROCT to see the robustness of the splits.



(a) multi-class OCT, accuracy = 0.837

(b) multi-class ROCT,ϵ = 0.05, accuracy = 0.747

Figure 5.2: Both models run on 300s runtime with the wine data set (van Rijn, 2014)

In Figure 5.2 we see that our robust formulation produces more robust splits: in 5.2 (a) notice that there are some splits that provide many opportunities for adversarial attacks, as portrayed in Figure 2.3. The multi-class OCT wins in accuracy over ROCT, but we see in the plot that this can be interpreted as over-fitting for accuracy.

5.3. Improving applicability and performance of ROCT

In this section we try to answer our research sub-question:

”Can the formulation of ROCT be changed or extended to make it more applicable and improve performance?”

Our results show that we can improve ROCT performance with shorter runtime by making changes to the formulation. This does not mean that ROCT is now suitable for large data sets, in this respect our out-performance is marginal. The formulation of ROCT is extended to apply to regression and multi-class classification tasks, which are common in practice.

6

Discussion and conclusion

6.1. Discussion

When we analyse our results in their entirety, we find that there is no simple solution to the trade-off between optimality and computational intensity. Current research and our work do not provide an algorithm with the runtime on the scale of CART and the optimality of OCT or ROCT. As far as robustness is concerned, the problem of a robust optimal classification tree is more complex than that of a normal optimal classification tree. Since there are no shortcuts for robust optimality yet, we quickly return to the use of heuristics.

Our results show different ways to achieve robustness, either optimal through the altered formulation or heuristically through data adaptations. We hope that these methods, and the insights gained from trying them out, will help the reader understand the difficulties related to the robustness of classification trees. And that our altered ROCT formulation enables simple faster optimisation in practical applications. As OCT research progresses and efficient Branch & Bound algorithms or MILP formulations are developed, our proposed restart algorithm can be used to balance robustness and accuracy.

We note that our definition of robustness has some limitations. In our report and most of the research, only l_∞ attacks are used. In reality, other norms or more abstract attacks can also be used. For example, an attacker could change the labels of certain samples (Rhuggenaath et al., 2018). This type of attack is difficult to implement in the OCT formulation without hyper parameters. We also did not evaluate the multi-class, regression OCT and restart OCT in detail. This was due to the time constraints that are unavoidable in a bachelor thesis. Another limitation of our research is that we forced the problem in MILP form. ILPs and MILPs are not made for fitting trees. Specialised methods can provide computational advantages. We note that regarding CART perturbations and the restart OCT, the non-robust nature of OCT and CART purely optimises accuracy, which is why it is hard to improve robustness without changing the formulation.

6.1.1. Recommendations

We hope that these limitations inspire new research, as it would be insightful to try different adversarial attacks and formulate MILPs for these attacks. We did not have time for trying other implementations such as constraint programming or creating our own Branch & Bound algorithm.

Current research only provides custom Branch & Bound algorithms for binary data sets. If the reader were to attempt to create such an algorithm, the difficulty would be to build optimal thresholds into the algorithm without exploding the number of branches in branch & bound.

There is still a need for research in the formulation of ROCT. We have not yet found a way to break the dominating symmetry, as is the case in Verwer and Zhang (2019) for OCT. This problem is more complex with ROCT because there is less symmetry in the s-variables, since multiple leaves can be reached by one sample. Finding a formulation that changes the exploding number of s variables would scale better.

Next, it would be interesting to examine different functions of the adversarial distance in a reweighing scheme applied to various models. Our method of increasing the powers shows the existing relationship but does not fully exploit it.

6.2. Conclusion

To answer our research question from section 1.1, this report tries to find simple and cheap robustness for classification trees. Several methods were investigated to make CART robust without changing the algorithm. We trained robust trees in multiple ways and made runtime improvements to an existing formulation for an optimal robust classification tree. We have also developed an algorithm that can be applied to any OCT formulation and extended an existing robust optimal classification tree MILP. To summarise our contributions:

- Investigated cheap robustness by adapting training data for CART with:
    1. Linear perturbations
    2. Uniform perturbations
    3. Adversarial examples
    4. Reweighing of training data
- Creation of an algorithm based on adversarial example generation to improve the robustness of any OCT formulation.
- Altered the ROCT formulation to make it faster to compute.
- Extended the ROCT formulation for regression and multi-class classification tasks.

The results of these contributions verify that non-robust cheap and simple heuristics are difficult to make robust without changing their formulation. Optimally robust decision trees are generated by ROCT, but they lack computational efficiency. We partially solve this problem by making ROCT compute faster.

Bibliography

Aghaei, S., Gómez, A., & Vayanos, P. (2021). Strong optimal classification trees. https://doi.org/10.48550/ARXIV.2103.15965

Bertsimas, J., D.Dunn. (2017). *Optimal classification trees*. Springer. https://doi.org/10.1007/s10994-017-5633-9

Blanquero, R., Carrizosa, E., Molero-Río, C., & Morales, D. R. (2021). Optimal randomized classification trees. *Computers & Operations Research*, *132*, 105281. https://doi.org/10.1016/j.cor.2021.105281

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1983). Classification and regression trees.

Chen, H., Zhang, H., Boning, D., & Hsieh, C.-J. (2019). Robust decision trees against adversarial examples. In K. Chaudhuri & R. Salakhutdinov (Eds.), *Proceedings of the 36th international conference on machine learning* (pp. 1122–1131). PMLR. https://proceedings.mlr.press/v97/19m.html

Feurer, M., van Rijn, J. N., Kadra, A., Gijsbers, P., Mallik, N., Ravi, S., Müller, A., Vanschoren, J., & Hutter, F. (2019). Openml-python: An extensible python api for openml. https://doi.org/10.48550/ARXIV.1911.02490

Firat, M., Crognier, G., Gabor, A. F., Hurkens, C., & Zhang, Y. (2020). Column generation based heuristic for learning classification trees. *Computers Operations Research*, *116*, 104866. https://doi.org/https://doi.org/10.1016/j.cor.2019.104866

Goodfellow, I. (2020). Attacking machine learning with adversarial examples. https://openai.com/blog/adversarial-example-research/

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

Justin, N., Aghaei, S., Gomez, A., & Vayanos, P. (2022). Optimal robust classification trees. *The AAAI-22 Workshop on Adversarial Machine Learning and Beyond*. https://openreview.net/forum?id=HbasA9ysA3

Kantchelian, A., Tygar, J. D., & Joseph, A. D. (2015). Evasion and hardening of tree ensemble classifiers. *CoRR*, *abs/1509.07892*. http://arxiv.org/abs/1509.07892

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *Journal of machine learning research*, *12*(Oct), 2825–2830.

Rhuggenaath, J., Zhang, Y., Akcay, A., Kaymak, U., & Verwer, S. (2018). Learning fuzzy decision trees using integer programming. *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 1–8. https://doi.org/10.1109/FUZZ-IEEE.2018.8491636

Szücs, D., & Schmidt, F. (2018). *Decision tree visualization for high-dimensional numerical data*.

van Rijn, J. (2014). Openml wine data set.

Verwer, S., & Zhang, Y. (2017). Learning decision trees with flexible constraints and objectives using integer optimization. https://doi.org/10.1007/978-3-319-59776-8_8

Verwer, S., & Zhang, Y. (2019). Learning optimal classification trees using a binary linear program formulation. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*(01), 1625–1632. https://doi.org/10.1609/aaai.v33i01.33011624

Vos, D., & Verwer, S. (2021a). 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). PMLR. https://proceedings.mlr.press/v139/vos21a.html

Vos, D., & Verwer, S. (2021b). Robust optimal classification trees against adversarial examples. https://arxiv.org/abs/2109.03857

A

Appendix

CART perturbation results

Adversarial accuracy

All results or different $N_{perturb}$ and p can be recreated with code in Github [1] under the ”Bachelor Thesis” repository.

Table A.1: Adversarial accuracy scores for all our perturbation schemes with $N_{perturb}$ = 20,50,20,20 respectively and p = 50 for the reweighing scheme.

name,ϵ,d	Linear perturb	Adv. examples	Reweighing scheme	Uniform perturb.
haberman,0.02,0	0.73	0.74	0.74	0.74
haberman,0.03,0	0.78	0.69	0.64	0.73
haberman,0.05,0	0.64	0.41	0.43	0.59
blood-transfusion,0.01,4	0.90	0.64	0.31	0.89
blood-transfusion,0.02,3	0.63	0.00	0.00	0.56
blood-transfusion,0.03,2	0.94	0.19	0.00	0.94
cylinder-bands,0.23,3	0.38	0.00	0.01	0.20
cylinder-bands,0.28,2	0.47	0.69	0.31	0.44
Use cylinder-bands,0.45,0	0.39	0.74	0.74	0.17
diabetes,0.05,0	0.54	0.69	0.51	0.69
diabetes,0.07,0	0.48	0.04	0.38	0.77
diabetes,0.09,0	0.63	0.64	0.30	0.57
ionosphere,0.2,2	0.66	0.00	0.00	0.82
ionosphere,0.28,2	0.27	0.62	0.00	0.46
ionosphere,0.36,1	0.40	0.00	0.10	0.51
banknote-authentication,0.07,4	0.70	0.69	0.31	0.62
banknote-authentication,0.09,3	0.25	0.74	0.74	0.30
banknote-authentication,0.11,3	0.89	0.68	0.85	0.88
breast-cancer,0.28,3	0.82	0.00	0.46	0.80
breast-cancer,0.39,2	0.50	0.64	0.29	0.82
breast-cancer,0.45,1	0.67	0.13	0.00	0.68
wine,0.02,1	0.68	0.03	0.98	0.71
wine,0.03,0	0.41	0.00	0.01	0.47
wine,0.04,2	0.00	0.63	0.69	0.00

[1] https://github.com/Gertlek

ROCT and Altered ROCT results

Table A.2: ROCT and altered ROCT run for 1800s

	(a) Adversarial accuracy		(b) Accuracy	
name,ϵ,d	**ROCT**	**ROCT2**	**ROCT**	**ROCT2**
haberman,0.02,0	0.741935	0.741935	0.741935	0.741935
haberman,0.03,0	0.741935	0.741935	0.741935	0.741935
haberman,0.05,0	0.741935	0.741935	0.741935	0.741935
blood-transfusion,0.01,4	0.9	0.906667	0.9	0.906667
blood-transfusion,0.02,3	0.9	0.906667	0.9	0.906667
blood-transfusion,0.03,2	0.906667	0.906667	0.906667	0.906667
cylinder-bands,0.23,3	0.571429	0.5	0.571429	0.535714
cylinder-bands,0.28,2	0.607143	0.535714	0.607143	0.553571
cylinder-bands,.36,1	0.517857	0.517857	0.517857	0.517857
diabetes,0.05,0	0.642857	0.642857	0.642857	0.642857
diabetes,0.07,0	0.642857	0.642857	0.642857	0.642857
diabetes,0.09,0	0.642857	0.642857	0.642857	0.642857
ionosphere,0.2,2	0.971831	0.971831	1	1
ionosphere,0.28,2	0.971831	0.971831	1	1
ionosphere,0.36,1	1	1	1	1
banknote-authentication, 0.07, 4	0.621818	0.589091	0.84	0.767273
banknote-authentication, 0.09, 3	0.276364	0.494545	0.378182	0.738182
banknote-authentication, 0.11, 3	0.247273	0.247273	0.374545	0.374545
breast-cancer,0.28,3	0.934307	0.934307	1	1
breast-cancer,0.39,2	0.875912	0.875912	0.948905	0.948905
breast-cancer,0.45,1	0.817518	0.817518	0.890511	0.890511
wine,0.02,1	0.702308	0.702308	0.730769	0.730769
wine,0.03,0	0.693077	0.693077	0.693077	0.693077
wine,0.04,2	0.693077	0.693077	0.693077	0.693077