Efficient Training of Robust Decision Trees Against Adversarial Examples

Daniël Vos, Sicco Verwer
Delft University of Technology
D.A.Vos@tudelft.nl, S.E.Verwer@tudelft.nl

Abstract

In the present day we use machine learning for sensitive tasks that require models to be both understandable and robust. Although traditional models such as decision trees are understandable, they suffer from adversarial attacks. When a decision tree is used to differentiate between a user’s benign and malicious behavior, an adversarial attack allows the user to effectively evade the model by perturbing the inputs the model receives. We can use algorithms that take adversarial attacks into account to fit trees that are more robust. In this work we propose an algorithm, GROOT, that is two orders of magnitude faster than the state-of-the-art while scoring competitively on accuracy against adversaries. GROOT accepts an intuitive and permissible threat model. Where previous threat models were limited to distance norms, we allow each feature to be perturbed with a user-specified parameter: either a maximum distance or constraints on the direction of perturbation. Previous works assumed that both benign and malicious users attempt model evasion but we allow the user to select which classes perform adversarial attacks. Additionally, we introduce a hyperparameter $\rho$ that allows GROOT to trade off performance in the regular and adversarial settings.

1 Introduction

Machine learning algorithms have been applied to a wide range of sensitive tasks in areas such as cyber security, health care and finance [Buczak and Guven, 2015; Chen et al., 2017]. In these tasks it is important that decisions made by the model are explainable [Samek et al., 2017] and that the model is robust against adversarial attacks. When we want to train a model to detect credit card fraud for example, we want to both understand how the model predicts fraud and we want to make it difficult for users to evade the model. Recently it has been shown that neural networks [Szegedy et al., 2014; Goodfellow et al., 2014] and similarly linear models, decision trees and support vector machines [Papernot et al., 2016a] are vulnerable to such evasions called adversarial examples: perturbed samples that trick the model into misclassifying them.

Many solutions have been proposed to increase the adversarial robustness of neural networks [Papernot et al., 2016b; Meng and Chen, 2017; Samangouei et al., 2018; Ilyas et al., 2019] but these models are inherently hard to explain. These solutions can also apply to linear classifiers, but such models perform poorly on data that is not linearly separable. Decision trees are a popular choice of explainable model and recently the first steps have been taken to train them robustly:

Kantchelian et al. [Kantchelian et al., 2016] proposed a hardening approach for decision tree ensembles using a mixed-integer linear programming approach and introduced an optimal attack. Chen et al. [Chen et al., 2019] trained robust trees by scoring with the worst-case information gain or Gini impurity criteria then proposed a fast approximation. Calzavara et al. [Calzavara et al., 2020] created TREANT, a flexible approach in which the user describes a threat model using axis-aligned rules and a loss function under attack that it directly optimizes.

In this work we propose GROOT, a new algorithm for training robust decision trees. Like Chen et al. [Chen et al., 2019], we closely mimic the greedy recursive splitting strategy that traditional decision trees use and we score splits with the worst-case Gini impurity. Additionally, we propagate perturbed samples to child nodes and search through more candidate splits. Since the adversarial Gini impurity is concave with respect to the number of modified data points, we can compute its value in constant time using an analytical solution. This makes GROOT two orders of magnitude faster than TREANT [Calzavara et al., 2020]. Moreover, it scores competitively on adversarial accuracy while accepting an intuitive and permissible threat model. Where previous works [Kantchelian et al., 2016; Chen et al., 2019; Calzavara et al., 2020] focus on robustness against all attacks, we recognize that models might be attacked in just a fraction of cases. We encode this in a parameter $\rho$ that trades off accuracy against regular users and robustness against attackers without influencing run-time. Our main contributions are:

- An efficient score function that allows us to fit trees two orders of magnitude faster than the state of the art.
- An algorithm that achieves competitive performance to the state of the art in the adversarial setting.
- A flexible implementation that allows users to specify attacks in terms of axis aligned perturbations.
2 Background and Related Work

2.1 Decision Tree Learning

A well-researched and interpretable model is the decision tree. Decision trees are comprised of nodes that perform a decision on a single feature of the dataset, sending some samples to a left child and others to the right. When we follow a path through these nodes we eventually reach a leaf that contains a prediction value. In this work we focus on two-class decision trees containing both decision nodes on numerical and categorical variables. Decision trees have recently gained more interest after their success in ensembles such as random forests and gradient boosting. Particularly approaches such as XGBoost [Chen and Guestrin, 2016] have dominated in recent machine learning competitions. It is worth noting that when used in large ensembles, the models lose their interpretability as the number of decision regions grows exponentially with the number of trees.

Popular methods for fitting decision trees greedily choose the best split according to the Gini impurity (CART [Breiman et al., 1984]) or information gain (ID3 [Quinlan, 1986]) criterion, then recursively continue this operation. This work focuses on the Gini impurity as the two criteria differ in their behavior only in 2% of the cases [Raileanu and Stoffel, 2004], but the information gain criterion requires an expensive logarithm operation to compute.

2.2 Adversarial Examples

When machine learning models are deployed in the real world, one can expect them to be attacked. One such attack is evasion, where a user will submit an input with specifically crafted perturbations so that it is misclassified. These perturbed inputs are known as ‘adversarial examples’ [Szegedy et al., 2014]. Generally, one assumes that an attacker can move samples within some limits surrounding the samples, e.g. a region of radius $\epsilon$ in the $L_\infty$ norm. In this work we allow users to specify these regions with a flexible specification and assume that the attacker knows everything about the model. This constitutes a white-box scenario as opposed to black-box in which the attacker can only use the model as an oracle.

2.3 Related Work

We briefly summarize three previous works in the area of adversarial learning decision trees. In Table 1 we compare each algorithm’s time complexity and threat model.

**Hardening Tree Ensembles**

Setting the foundations of robust decision trees, Kantchelian et al. [Kantchelian et al., 2016] propose a hardening approach for tree ensembles. They show that finding adversarial examples under distance constraints is NP-hard for tree ensembles and they provide a MILP translation of the problem for arbitrary $L_x$ norm. Using MILP for hardening is complete but inefficient, they therefore also give an approximation algorithm for attacking a model under the $L_0$ norm constraints. Although hardening using the approximate attack shows an increase in classifier robustness under $L_0$ norm attacks, the classifier performs worse under different norms. This highlights that the threat model during training needs to closely represent the threat model at test time. The hardening approach can not be used to fit a single robust tree.

**Robust Decision Trees Against $L_\infty$ Attacks**

Chen et al. [Chen et al., 2019] assume an attacker that is restricted by an $L_\infty$ norm and give an algorithm to fit a tree that is robust to these attacks by using the ‘information gain under attack’. This criterion is the information gain for a situation in which the attacker moves points within an $L_\infty$ radius to confuse the classifier maximally. In general terms Chen et al.’s algorithm finds the best split, moves points near the split over it if that reduces the model’s information gain and recursively continues this splitting procedure on the left and right side of the split. This algorithm closely follows the greedy tree building approach for normal decision trees and intuitively increases robustness against an $L_\infty$ threat model.

**TREANT**

TREANT [Calzavara et al., 2020] introduces a more flexible approach to specifying attacker capabilities. By allowing the user to describe an adversary using axis-aligned rules, attackers can be more realistically modelled with asymmetric changes and different constraints for different axes. Also, attackers can be modelled with a ‘budget’ that they can spend on changing data points which allows the user to evaluate robustness against attackers of different strengths. TREANT directly optimizes a loss function instead of using a splitting criterion like in regular or robust decision trees [Chen et al., 2019]. Although this allows TREANT to train robust models against a variety of attackers, their algorithm deploys a solver to optimize the loss function and pre-computes all possible attacks in exponential time.

3 Specifying Threat Models

In our work we assume existence of an attacker that knows the decision tree and perturbs samples according to a user-specified threat model. Therefore to support a wide range of attack types we take inspiration from TREANT [Calzavara et al., 2020] and let the user define the perturbation limits for each individual feature. The specification is as follows:

- ‘-or None: This feature cannot be perturbed.
- > or <: This feature can be increased / decreased.
- <>: This feature can be perturbed to any value.

| Algorithm       | Time complexity | Threat model       |
|-----------------|-----------------|--------------------|
| GROOT (ours)    | $O(n \log n 2^d f)$ | $L_\infty$ and variations |
| TREANT          | $O(n^2 2^d d^2 f)$ | axis-aligned rules |
| Robust trees    | $O(n^2 2^d f)$   | $L_\infty$         |
| MILP hardening  | $O(n \log n X 2^d f)$ | $L_0 / L_1 / L_2 / L_\infty$ |
| Approx. hardening| $O(n \log n i d^2 f)$ | $L_0$              |
| Regular trees   | $O(n \log n 2^d f)$ | –                  |

Table 1: Overview of algorithms for fitting robust decision trees. Where $n$ is the number of samples, $f$ number of features, $d$ tree depth, $r$ number of axis-aligned rules, $X$ solver time and $i$ hardening iterations. Recent works move away from $L$ norms to support more flexible specifications.
The feature can be perturbed by a distance of \( \epsilon \).

• \((\epsilon_l, \epsilon_r)\): The feature can be perturbed \( \epsilon_l \) left or \( \epsilon_r \) right.

It is worth noting that all these cases can be translated to the tuple notation, e.g., we can encode \( > \) as \((0, \infty)\) or a number \( \epsilon \) as \((\epsilon, \epsilon)\). For conciseness we only use the tuple notation in the algorithms in section 5. When we set the threat model to \( \epsilon \) for each feature, it behaves identically to an \( L_\infty \) norm. In that case GROOT performs similarly to the Gini impurity variant of algorithm 1 by Chen et al. [Chen et al., 2019], with only changes to the number of candidate splits and what samples propagate left and right. Our implementation of GROOT and adversarial attacks can be found on GitHub\(^1\).

In this work we assume that we train models to distinguish between two classes. In contrast to previous works we allow one to select whether the attacker can move all samples or only samples of one class. We do this since in secure settings it would intuitively be impractical for a user’s benign data to be predicted as malicious. Take for example a spam email detector, a benign user would not want their emails to be predicted as spam while a spammer would want their emails to be mistaken as benign. We give examples of threat models and highlight their importance in Figure 1.

---

1[https://github.com/tudelft-cda-lab/GROOT](https://github.com/tudelft-cda-lab/GROOT)
We can then write the $x'$ and $y'$ that maximize it as:

$$x', y' = \arg \max_{x \in [a, b]} \left( \frac{(L_0 + a)(L_1 + x)}{L_0 + L_1 + x + y} + \frac{(R_0 + I_0 - y)(R_1 + I_1 - x)}{R_0 + R_1 + I_0 + I_1 - x - y} \right)$$

The algorithm by Chen et al. [Chen et al., 2019] optimizes a similar function by iterating through the $I$ samples to perform gradient ascent. However, since the function is concave with respect to $x$ and $y$, we can do this faster by maximizing the function analytically and rounding to a near integer solution. The maxima form the following line (proofs in the appendix):

$$y' = \frac{L_1(R_0 + I_0) - L_0(R_1 + I_1)}{L_1 + R_1 + I_1} + \frac{(L_0 + R_0 + I_0)x'}{L_1 + R_1 + I_1}$$

(5)

Using gradient ascent and given enough movable samples one would end up on the optimal line but not necessarily on the closest point on the line to the starting values for $x$ and $y$. We argue that the closest point is intuitively a better solution than a random point on the line, because it represents the least number of samples to move. Therefore in GROOT we find the closest point on the solution line to the starting values $x, y$ then round to the nearest integers.

Computing the point and rounding it all takes $O(1)$ time. Therefore we have an efficient method (constant time) to compute $S_{\text{robust}}$.

### 4.2 Adversarial Gini Impurity for Malicious Samples

In GROOT we intend to allow users to choose whether all samples can move or only malicious samples. When only malicious samples move, computing the adversarial Gini impurity becomes simpler. In that case we can rewrite Equation 5 in terms of $y$, then set $y$ and $I_0$ to 0 since benign samples do not move:

$$x' : \frac{L_0 R_1 + L_0 I_1 - L_1 R_0}{L_0 + R_0}$$

(6)

This solution can again be computed in constant time then rounded to $\lfloor x' \rfloor$ and $\lceil x' \rceil$ to find the integer maximum of $x$.

### 5 GROOT

We introduce GROOT (Growing ROBust Trees), an algorithm that trains decision trees that are robust against adversarial examples generated from a user-specified threat model. The algorithm stays close to regular decision tree learning algorithms but searches through more candidate splits with a robust score function and propagates samples according to an attacker. Like regular decision tree learning algorithms, GROOT runs in $O(n \log n)$ time in terms of $n$ samples.

### 5.1 Scoring Candidate Splits

Similar to regular decision tree learning algorithms we can search over all possible splits and compute a score function to find the best split. In Algorithm 1 we iterate over each sample in sorted order to identify candidate splits. We evaluate each candidate split with the adversarial Gini impurity from the previous subsection to find the split that is most accurate against an adversary. To support both numerical and categorical features we split the search up in the two cases and describe each of these below. The time complexity in terms of $n$ samples for both cases is bounded by $O(n \log n)$ per feature.

#### Numerical case

In regular decision tree learning algorithms we score candidate splits at each position in which a sample moves from the right to the left side. However, when an adversary can perturb samples there are more possible splits that affect the sample counts on each side. Therefore we consider also candidate splits where a movable sample becomes in or out of range of $I$. Take for example a sample at position $3.3$ that can be perturbed in a radius of $0.2$, we score a split at $3.1$, $3.3$ and $3.5$. At the start of Algorithm 1 we sort all candidate splits and store what will happen to the counts $L / R / I$ at those splits, e.g. a sample moves from $R$ to $R I$. We can evaluate splits in $O(1)$ time as explained in Section 4 and we consider at maximum $3n$ splits, where $n$ is number of samples. Therefore the time complexity of evaluating splits is $O(n)$ per feature and this means the fitting run time is dominated by the sorts of complexity $O(n \log n)$.

#### Categorical case

To support categorical variables, we must find the best partition of categories with respect to the scoring function. Although a linear time algorithm (in number of categories) exists for fitting regular decision trees [Coppersmith et al., 1999], we have not found such an algorithm for robust decision trees. Since the number of categories in a feature is usually bounded by a small constant, we perform an exponential time search over all possible partitions. We can count the number of samples of each category in linear time, so the complexity for this procedure is $O(n2^n)$ where $n$ is the number of samples and $c$ the number of categories. We do not explicitly give an algorithm as it follows the same idea as Algorithm 1.

### 5.2 Propagating Samples

When fitting regular decision trees one can simply move all samples lower than a threshold left and higher to the right. To account for samples that the adversary moves we make a modification to this propagation as we define in Algorithm 2. We do not only keeping track of left ($L$) and right ($R$) samples, but also store ‘intersection’ sets ($RI, LI$) that contains samples that can move to both sides.

In section 4 we showed that $x'$ and $y'$ are the optimal values for the adversarial Gini impurity. Given $x'$ we move samples from $I_1$ over the split to place $x'$ samples on the left and $I_1 - x'$ on the right. If there were fewer than $x'$ samples on the left we move samples from the right, if there were more samples on the left we move them to the right. The actual samples that move are randomly selected from the intersection. If the user selects that the attacker can move all classes we repeat this operation for $I_0$ and $y'$ to account for moving benign samples.
Algorithm 1 Find Best Robust Split on Numerical variable

1: function BESTROBUSTSPLIT(X)
2:  \( S \leftarrow X \cup \{ o | o \in X \land o > s + \varepsilon_r \} \)
3:  \( R \leftarrow \{ o | o \in X \land o < s + \varepsilon_r \} \)
4:  \( L \leftarrow \{ o | o \in X \land s - \varepsilon_l \leq o \leq s \} \)
5:  \( L_0 \leftarrow L_0 + (1 - \rho)\{ L_0 \} \)
6:  \( l_1 \leftarrow l_1 + (1 - \rho)\{ L_1 \} \)
7:  \( r_0 \leftarrow r_0 + (1 - \rho)\{ R_0 \} \)
8:  \( r_1 \leftarrow r_1 + (1 - \rho)\{ R_1 \} \)
9:  \( i_0 \leftarrow \rho\{ R_0 \} + \rho LI_0 \), \( i_1 \leftarrow \rho\{ R_1 \} + \rho LI_1 \)
10: \( (x_s, y_s) \leftarrow \text{close point on Eq. 5} \) to \( (LI_0, LI_1) \)
11: \( x_s \leftarrow |x_s| \), \( y_s \leftarrow |y_s| \)
12: \( g_s \leftarrow S(l_0 + y_s, l_1 + x_s, r_0 + i_0 - y_s, r_1 + i_1 - x_s) \)
13: end function

Algorithm 2 Fit Robust Tree on Numerical Data

1: function FITROBUSTTREE(X)
2: if stopping criterion (e.g. maximum depth) then
3:  return Leaf(\(|X_0|, |X_1|\))
4: else
5:  for \( f \leftarrow 1...F \) do
6:    \( s_f, g_f, x_f \leftarrow \text{BESTROBUSTSPLIT}(X^f) \)
7:  end for
8: \( f' \leftarrow \text{arg min}_f g_f \)
9: determine \( R, R, L, L \) for split \( f' \)
10: move \( (1 - \rho)\{ L \} \) samples from \( L \) to \( L \)
11: move \( (1 - \rho)\{ R \} \) samples from \( R \) to \( R \)
12: if \( |L| < |x_{f'}| \) then
13:  move \( |L| - |x_{f'}| \) samples from \( R \) to \( L \)
14: else
15:  move \( x_{f'} - |L| \) samples from \( L \) to \( R \)
16: end if
17: node \( \leftarrow \text{FITROBUSTTREE}(L \cup L) \)
18: node \( \leftarrow \text{FITROBUSTTREE}(R \cup R) \)
19: return DecisionNode(\( s_{f'}, node_l, node_r \))
20: end function

5.3 Trading Off Regular and Adversarial Performance

When maximizing performance in an adversarial setting one sacrifices performance in the regular unperturbed setting. To trade off between performance in these settings, we introduce a parameter \( 0 \leq \rho \leq 1 \) that determines the fraction of malicious points that can move. Intuitively this encodes what percentage of attackers attempt an evasion attack. For example, if we deploy a robust model for malware detection, we can expect that only a fraction of malware creators will change their code to evade the model.

We make two changes to the algorithms to account for \( \rho \). First, we scale the variable \( i \) in Algorithm 1 to \( \rho i \). Second, during sample propagation (Algorithm 2) we reduce the size of the set \( I \) to \( \rho |I| \) by randomly returning \( (1 - \rho) |I| \) samples to their unperturbed side of the split (\( L \) or \( R \)).

6 Results

We give results of regular (natural) decision trees, TREANT [Calzavara et al., 2020], Chen et al. [Chen et al., 2019] and GROOT on fourteen datasets in which we compare predictive performance and run time. For natural decision trees we use scikit-learn’s [Pedregosa et al., 2011] implementation as it is widely used for research in the field. To compare against robust decision tree algorithms we run TREANT and the heuristic from Chen et al. that computes the adversarial Gini impurity using four representative cases. All datasets can be retrieved from openML except breast-cancer, their specific versions are listed in the tables. Breast-cancer was imported from Scikit-learn.

6.1 Predictive performance

To determine the quality of the trees produced by each algorithm we measure predictive performance in a regular and adversarial setting. When measuring in the adversarial setting, an attacker first optimally moves all test samples within the perturbation limits to maximize the number of misclassifications. To maximize the number of misclassifications within these limits we loop over all tree leaves and check if the leaf is in reach of the sample. If any reachable leaf predicts a different class than the actual label we count a misclassification. The adversarial accuracy is the accuracy after this procedure or equivalently the success rate of an attacker given the perturbation limits. In the experiments we test against an attacker that can move samples freely within \( L_\infty \) distance \( \epsilon = 0.1 \).

We encoded the above threat models in TREANT’s attack rules using precondition \([−\infty, \infty]\) and postcondition \( \epsilon \) or \(-\epsilon\). Each rule has cost 1 and the attacker has a budget equal to the depth of the trees. To exactly match the threat model specifications, we modify TREANT to only use attack rules once per feature. Preliminary testing without this modification gave poor results. Given these modifications the attack rules exactly encode the \( L_\infty \) radius attack model.

A popular method for measuring robustness (e.g. in [Chen et al., 2019]) is to compute the average perturbation distance required to cause a misclassification. We choose against this metric as it assumes features can perturb arbitrarily and with equal cost.

We train a decision tree on each dataset with 5 fold stratified cross validation. All algorithms were trained up to a depth of 4 to maintain interpretability. We implemented GROOT and the heuristic by Chen et al. in python but used TREANT’s existing implementation with minor modifications. Scikit-learn uses the Gini impurity and TREANT optimizes the sum of squared errors. We used \( \rho = 1 \) for GROOT to maximize adversarial performance. We used the default hyperparameter settings from Scikit-learn for all models which means leaves needed 2 samples to be created and trees could split multiple times on the same feature. We present results on adversarial accuracy in Table 2 and sum-
Table 2: Test accuracy scores after adversarial attacks, best adversarial scores are bold. Mean and standard deviations were computed over 5 fold cross validation. An adversarial attack succeeded when the decision tree had a leaf of different prediction label within $L_{\infty}$ distance 0.1.

| Dataset                        | Samples | Features | Chen et al. | GROOT | Natural | TREANT |
|-------------------------------|---------|----------|-------------|-------|---------|--------|
| banknote-authentication (1)   | 1372    | 4        | .929 ± .020 | .943 ± .017 | .930 ± .034 | .938 ± .027 |
| blood-transfusion (1)         | 748     | 4        | .769 ± .033 | .769 ± .033 | .769 ± .033 | .762 ± .012 |
| breast-cancer                 | 569     | 30       | .926 ± .013 | .926 ± .013 | .926 ± .013 | .926 ± .014 |
| climate-model-simulation (4) | 540     | 18       | .844 ± .044 | .902 ± .012 | .748 ± .041 | .915 ± .004 |
| cylinder-bands (2)            | 277     | 37       | .664 ± .076 | .664 ± .076 | .603 ± .037 | .621 ± .085 |
| diabetes (1)                  | 768     | 8        | .729 ± .025 | .727 ± .016 | .720 ± .020 | .732 ± .029 |
| haberman (1)                  | 306     | 3        | .722 ± .046 | .722 ± .046 | .722 ± .046 | .706 ± .038 |
| ionosphere (1)                | 351     | 34       | .880 ± .016 | .872 ± .010 | .712 ± .048 | .869 ± .031 |
| parkinsons (1)                | 195     | 22       | .800 ± .066 | .826 ± .046 | .174 ± .084 | .821 ± .065 |
| planning-relax (1)            | 182     | 12       | .655 ± .067 | .676 ± .043 | .468 ± .091 | .709 ± .013 |
| sonar (1)                     | 208     | 60       | .336 ± .096 | .432 ± .083 | .049 ± .096 | .500 ± .073 |
| spambase (1)                  | 4601    | 57       | .843 ± .014 | .874 ± .019 | .340 ± .061 | .837 ± .014 |
| SPECTF                        | 267     | 44       | .753 ± .059 | .753 ± .059 | .749 ± .037 | .719 ± .058 |
| wine (1)                      | 6497    | 11       | .635 ± .043 | .681 ± .012 | .500 ± .057 | .670 ± .016 |

Figure 3: Average accuracy scores over 14 datasets before and after adversarial attacks. All models fit accurate trees while TREANT and GROOT perform best in the adversarial setting.

Figure 4: Runtimes in seconds on a logarithmic scale for fitting decision trees of depth 4. Natural trees use Scikit-learn’s fast implementation, Chen et al. and GROOT ran up to a thousand times faster than TREANT.

While all four models considered scored well on accuracy, the scores in the adversarial setting differ. TREANT and GROOT perform similarly on adversarial accuracy and both significantly improve on Scikit-learn’s natural decision tree scores by approximately 21%. The heuristic by Chen et al. score approximately 2% worse than TREANT and GROOT. The cases where TREANT outperforms GROOT and Chen could be explained by TREANT’s attack invariance that prevents deeper trees from becoming less robust. The heuristic by Chen et al. finds an improvement over GROOT for one dataset with a 0.8% improvement in adversarial accuracy. We expect this result to be insignificant as it is within the range of one standard deviation.

On the blood-transfusion and haberman datasets the natural decision trees perform equally well on adversarial accuracy as the robust tree fitting algorithms. These datasets can be well predicted using features that have samples that lie far apart i.e. more than 0.1 distance.

6.2 Run time

To compare the efficiency of the algorithms, we also recorded the run times in Figure 4. All experiments ran on a linux machine with 16 Intel Xeon CPU cores and 72GB of RAM total. Each algorithm instance ran on a single core.

Although the run time complexities for scikit-learn and our work are similar, the efficient implementation of scikit-learn runs fast even for large sample sizes. It is worth noting that GROOT cannot be faster than scikit-learn as GROOT searches over more candidate splits. Comparing TREANT and GROOT, we see that our algorithm runs on average two
orders of magnitude faster. TREANT exhaustively searches for attacks using an exponential search and uses a sequential quadratic programming solver to optimize the loss function which likely contributes to the higher run time. The heuristic by Chen et al. has a similar runtime as GROOT as it is implemented in the same python code and shares the same time complexity.

7 Discussion and conclusions

We present GROOT, a greedy algorithm for learning robust decision trees. It uses an analytical solution for computing the adversarial Gini impurity and runs two orders of magnitude faster than existing approaches. Our results show that GROOT runs in a matter of seconds and is competitive with the state-of-the-art TREANT method.

An interesting direction for future work is to investigate the performance of GROOT in a black box setting. Specifically, we are interested in the susceptibility to boundary and transfer attacks. As GROOT was designed for classification between benign and malicious data points it currently does not support regression tasks. However, decision tree learners can be modified to work on continuous target values. We aim to investigate such modifications in future work, as well as the performance in random forests and gradient boosting ensembles.

For the sake of comparison, we experimented using the same public datasets as similar works on robustness, but these datasets were originally not intended for research into adversarial attacks. With GROOT’s flexible implementation, it becomes possible to model different kinds of attack on different kinds of datasets, bringing adversarial machine learning studies closer to the real world. In the near future, we will apply this framework to datasets from problems where adversarial modifications are an important concern such as fraud and intrusion detection.

We conclude that:

- By solving the adversarial Gini impurity analytically we can now fit robust trees with the same time complexity as regular trees: $O(n \log n)$, for $n$ samples.
- This algorithm runs two orders of magnitude faster than the state-of-the-art work and as efficiently as a fast heuristic.
- GROOT consistently achieves scores competitive with the state-of-the-art work in terms of robustness.

References

[Breiman et al., 1984] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. Classification and Regression Trees. Wadsworth and Brooks, Monterey, CA, 1984.

[Buczak and Guven, 2015] Anna L Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications surveys & tutorials*, 18(2):1153–1176, 2015.

[Calzavara et al., 2020] Stefano Calzavara, Claudio Lucchesi, Gabriele Tolomei, Seyum Assefa Abebe, and Salvatore Orlando. Treant: Training evasion-aware decision trees. *Data Mining and Knowledge Discovery*, pages 1–31, 2020.

[Chen and Guestrin, 2016] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[Chen et al., 2017] Min Chen, Yixue Hao, Kai Hwang, Lu Wang, and Lin Wang. Disease prediction by machine learning over big data from healthcare communities. *Ieee Access*, 5:8869–8879, 2017.

[Chen et al., 2019] Hongge Chen, Huan Zhang, Duane Boning, and Cho-Jui Hsieh. Robust decision trees against adversarial examples. In *ICML*, pages 1122–1131, 2019.

[Coppersmith et al., 1999] Don Coppersmith, Se June Hong, and Jonathan RM Hosking. Partitioning nominal attributes in decision trees. *Data Mining and Knowledge Discovery*, 3(2):197–217, 1999.

[Goodfellow et al., 2014] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[Ilyas et al., 2019] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. In *Advances in Neural Information Processing Systems*, pages 125–136, 2019.

[Kantchelian et al., 2016] Alex Kantchelian, J Doug Tygar, and Anthony Joseph. Evasion and hardening of tree ensemble classifiers. In *ICML*, pages 2387–2396, 2016.

[Meng and Chen, 2017] Dongyu Meng and Hao Chen. Mag-net: a two-pronged defense against adversarial examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 135–147, 2017.

[Papernot et al., 2016a] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016.

[Papernot et al., 2016b] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597. IEEE, 2016.

[Pedregosa et al., 2011] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2387–2396, 2011.

[Quinlan, 1986] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.
[Raileanu and Stoffel, 2004] Laura Elena Raileanu and Kilian Stoffel. Theoretical Comparison between the Gini Index and Information Gain Criteria. *Annals of Mathematics and Artificial Intelligence, 41*(1):77–93, May 2004.

[Samangouei et al., 2018] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-GAN: Protecting classifiers against adversarial attacks using generative models. In *International Conference on Learning Representations*, 2018.

[Samek et al., 2017] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv preprint arXiv:1708.08296*, 2017.

[Szegedy et al., 2014] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna Estrach, Dumitru Erhan, Ian Goodfellow, and Robert Fergus. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014*, 2014.
Appendix: Adversarial Gini Impurity Proofs and Solutions

In Section 4 we introduced the adversarial Gini impurity, a score function that promotes robust splits. This function consists of a maximization term that is dependent on two variables: $x$ and $y$. Although we identified the analytical solutions for $x$ and $y$ we did not prove they were optimal or show how to find the solutions. To show the solutions are optimal we first prove that the maximization term of the adversarial Gini impurity is concave with respect to $x$ and $y$. Given that this term is concave we then find the maxima by identifying the critical points that fall inside our region of interest.

Proof of Concavity

Let us prove that the maximization term of the adversarial Gini impurity is concave so that critical points are maxima. The maximization term is:

$$f(x, y) = \frac{(L_0 + y)(L_1 + x)}{L_0 + L_1 + x + y} + \frac{(R_0 + I_0 - y)(R_1 + I_1 - x)}{R_0 + R_1 + I_0 + I_1 - x - y}$$

We can get an intuition for $f$’s concavity by plotting it for some random choices of $L$, $R$ and $I$, see Figure 5.

A function is concave if its hessian is negative semi definite. To show that the hessian is negative semi definite we can use Sylvester’s criterion. This criterion states that if the even order principal minors of a hermitian matrix are non-negative and the odd order principal minors non-positive then the matrix is negative semi definite. Since the hessian of a real function with two variables $(x, y)$ is symmetric and real, it is hermitian. Therefore we can apply Sylvester’s criterion here. Let us first compute the hessian:

$$H = \begin{bmatrix} \frac{\partial^2}{\partial x^2} f(x, y) & \frac{\partial^2}{\partial x \partial y} f(x, y) \\ \frac{\partial^2}{\partial y \partial x} f(x, y) & \frac{\partial^2}{\partial y^2} f(x, y) \end{bmatrix}$$

Let us work out each term:

$$\frac{\partial}{\partial x} \left( \frac{\partial}{\partial x} f(x, y) \right) = -2 \frac{L_0 + y}{(L_0 + L_1 + x + y)^2} + \frac{(L_0 + y)(L_1 + x)}{(L_0 + L_1 + x + y)^3} - \frac{(R_0 + R_1 + I_0 + I_1 - x - y)^2}{(R_0 + R_1 + I_0 + I_1 - x - y)^3}$$

$$\frac{\partial}{\partial y} \left( \frac{\partial}{\partial y} f(x, y) \right) = -2 \frac{L_0 + x}{(L_0 + L_1 + x + y)^2} + \frac{(L_0 + x)(L_1 + x)}{(L_0 + L_1 + x + y)^3}$$

Since the order of differentiation does not matter, we have that: $\frac{\partial}{\partial x} \left( \frac{\partial}{\partial y} f(x, y) \right) = \frac{\partial}{\partial y} \left( \frac{\partial}{\partial x} f(x, y) \right)$. Therefore we only have to compute one of the sides of this equation.

$$\frac{\partial}{\partial x} \left( \frac{\partial}{\partial y} f(x, y) \right) = \frac{\partial}{\partial y} \left( \frac{\partial}{\partial x} f(x, y) \right) = 1 \frac{\partial}{\partial x} f(x, y) + \frac{\partial}{\partial y} f(x, y) \frac{\partial^2}{\partial x \partial y} f(x, y)$$

$$= \frac{\partial}{\partial x} \left( \frac{\partial}{\partial y} f(x, y) \right) + \frac{\partial}{\partial y} \left( \frac{\partial}{\partial x} f(x, y) \right) \frac{\partial^2}{\partial x \partial y} f(x, y)$$

$$= \frac{\partial^2}{\partial x^2} f(x, y) + \frac{\partial^2}{\partial x \partial y} f(x, y) + \frac{\partial^2}{\partial y \partial x} f(x, y) + \frac{\partial^2}{\partial y^2} f(x, y)$$

$$= \frac{\partial^2}{\partial x^2} f(x, y) + \frac{\partial^2}{\partial x \partial y} f(x, y) + \frac{\partial^2}{\partial x \partial y} f(x, y) + \frac{\partial^2}{\partial y^2} f(x, y)$$

Now let us refer back to Sylvester’s criterion, we want to show that the first principal minor is non-positive and the second non-negative. The first principal minor is simply the term in the top-left corner of $H$:

$$-2 \frac{(L_0 + y)^2}{(L_0 + L_1 + x + y)^3} - 2 \frac{(R_0 + I_0 - y)^2}{(R_0 + R_1 + I_0 + I_1 - x - y)^3}$$

Since the terms $L_0, L_1, R_0, R_1$ and $I_0 - x$, $I_0 - y$ are all non-negative, both fractions are non-negative. The $-2$ in front of them then make the both non-positive. Therefore for our region of interest the first principal minor is non-positive.

Now we continue with the second principal minor, this is the determinant of $H$. For simplicity let us label the four elements of $H$:

$$H = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

The second principal minor is then $\det H = ad - bc$. We will show that this determinant is positive:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} \frac{(L_0 + y)^2}{(L_0 + L_1 + x + y)^3} & \frac{(R_0 + I_0 - y)^2}{(R_0 + R_1 + I_0 + I_1 - x - y)^3} \\ \frac{(L_0 + L_1 + x + y)^2}{(L_0 + L_1 + x + y)^3} & \frac{(R_0 + R_1 + I_0 + I_1 - x - y)^2}{(R_0 + R_1 + I_0 + I_1 - x - y)^3} \end{bmatrix}$$

Because of the square, the numerator is always non-negative. Since $x \leq I_1$ and $y \leq I_0$ the denominator is also always non-negative. Therefore the second principal minor is non-negative.

Since the first principal minor is non-positive and the second principal minor is non-negative, the hessian is negative semi-definite. Since the hessian is negative semi-definite the function is concave with respect to $x$ and $y$. 


Finding the Maxima

To find the maximum of the function $f(x, y)$ we first find all critical points by setting both the partial derivative with respect to $x$ and the partial derivative with respect to $y$ equal to zero:

$$
\begin{align*}
\frac{\partial}{\partial x} \left( \frac{(L_0+y)(L_1+x)}{L_0+L_1+x+y} + \frac{(R_0+I_0-y)(R_1+I_1-x)}{R_0+R_1+I_0+I_1-x-y} \right) &= 0 \\
\frac{\partial}{\partial y} \left( \frac{(L_0+y)(L_1+x)}{L_0+L_1+x+y} + \frac{(R_0+I_0-y)(R_1+I_1-x)}{R_0+R_1+I_0+I_1-x-y} \right) &= 0
\end{align*}
$$

This system of equations has three solutions that we will examine, only one of which lies in our region of interest ($x \in [0, I_1], y \in [0, I_0])$:

- $x = -L_1, y = -L_0$: For any value of $L_0$ and $L_1$ other than 0 the solution will fall outside of the ranges defined for $x$ and $y$.
- $x = R_1 + I_1, y = R_0 + I_0$: For any value of $R_0$ and $R_1$ other than 0 the solution will fall outside of the ranges defined for $x$ and $y$.
- $x$ : free, $y = \frac{L_1 R_0 - L_0 R_1 - L_0 I_1 + L_1 I_0 + (L_0 + R_0 + I_0)x}{L_1 + R_1 + I_1}$.

These solutions lie on a line through our region of interest.

So now we have found a line that lies in our region of interest and consists of critical points:

$$
y = \frac{L_1 R_0 - L_0 R_1 - L_0 I_1 + L_1 I_0 + (L_0 + R_0 + I_0)x}{L_1 + R_1 + I_1}
$$

In the previous section we proved $f(x, y)$ was in fact concave with respect to $x$ and $y$. Therefore this line of critical points is a line of maxima of the function.

Integer Solutions

Instead of setting $x$ and $y$ to zero, we derive a line through the diagonals of the plots that is closest to the starting values of $x$ and $y$ ($LI_0, LI_1$) but we did not explain how to compute this point. To efficiently compute the closest point on the line we can use the fact that it lies on a second line perpendicular to the first line. So we find the perpendicular line that passes through $(LI_0, LI_1)$ and compute its intersection with our first line. This intersection is $(x', y')$. We can first write the solution line we found in a more common format:

$$
\frac{L_0 + R_0 + I_0}{L_1 + R_1 + I_1} x - y + \frac{L_1 R_0 - L_0 R_1 - L_0 I_1 + L_1 I_0}{L_1 + R_1 + I_1} = 0
$$

The point on this line closest to $(LI_0, LI_1)$ is then:

$$
x' = \frac{-(-LI_0 - \frac{L_0 + R_0 + I_0}{L_1 + R_1 + I_1} I_1)}{\left( \frac{L_0 + R_0 + I_0}{L_1 + R_1 + I_1} \right)^2 + 1} - \frac{L_0 + R_0 + I_0}{L_1 + R_1 + I_1} \frac{L_1 R_0 - L_0 R_1 - L_0 I_1 + L_1 I_0}{L_1 + R_1 + I_1}
$$

$$
y' = \frac{\frac{L_0 + R_0 + I_0}{L_1 + R_1 + I_1}}{\left( \frac{L_0 + R_0 + I_0}{L_1 + R_1 + I_1} \right)^2 + 1} \left( LI_0 + \frac{L_0 + R_0 + I_0}{L_1 + R_1 + I_1} LI_1 \right) + \frac{L_1 R_0 - L_0 R_1 - L_0 I_1 + L_1 I_0}{L_1 + R_1 + I_1}
$$

Whenever we want to compute $S_{1,\text{closest}}$ we can now in constant time determine $x', y'$ and round them to the nearest integers. If $x'$ or $y'$ happen to lie outside of the regions $[0, I_1]$ and $[0, I_0]$ respectively, we can clip their values to the boundaries.