Efficient Training of Robust Decision Trees Against Adversarial Examples

Daniël Vos [1] **Sicco Verwer** [1]

Abstract

Current state-of-the-art algorithms for training robust decision trees have high runtime costs and require hours to run. We present GROOT, an efficient algorithm for training robust decision trees and random forests that runs in a matter of seconds to minutes. Where before the worst-case Gini impurity was computed iteratively, we find that we can solve this function analytically to improve time complexity from $\mathcal{O}(n)$ to $\mathcal{O}(1)$ in terms of n samples. Our results on both single trees and ensembles on 14 structured datasets as well as on MNIST and Fashion-MNIST demonstrate that GROOT runs several orders of magnitude faster than the state-of-the-art works and also shows better performance in terms of adversarial accuracy on structured data.

1. Introduction

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 adversarial examples: perturbed samples that trick the model into misclassifying them. Much research has gone into training robust neural networks (Papernot et al., 2016b; Meng & Chen, 2017; Samangouei et al., 2018; Ilyas et al., 2019). These models perform well on unstructured data such as images and audio, but decision tree ensembles often outperform them on structured data. Additionally, when using a single decision tree, the models are easily interpreted by humans. Recently the first methods have been proposed to train decision trees and their ensembles robustly (Kantchelian et al., 2016; Chen et al., 2019; Calzavara et al., 2020; Andriushchenko & Hein, 2019) but the state-of-the-art methods are expensive to run.

In this work we propose GROOT, an efficient 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 adversarial Gini impurity. We prove that the adversarial Gini impurity is concave with respect to the number of modified data points and use its analytical solution to compute the function in constant time. Our results show that GROOT trains trees 3 to 6 orders of magnitude faster than the state-of-the-art method TREANT (Calzavara et al., 2020) and GROOT trains random forests 100-1000 times faster than provably robust boosting (Andriushchenko & Hein, 2019). Leveraging this speedup we can fit robust random forests using the adversarial Gini impurity and we do not have to rely on a heuristic such as Chen et al.

Moreover, GROOT scores competitively on adversarial accuracy which we evaluate on 14 structured datasets as well as on MNIST and Fashion-MNIST. On the structured data, both GROOT trees and GROOT forests outperform the state-of-the-art robust tree and forest methods. GROOT trees obtain a small performance improvement over TREANT. GROOT forests outperform provably robust boosting. Interestingly, GROOT trees and forests obtain top and similar ranks. Showing that in contrast to regular accuracy, there is not much difference between the adversarial accuracy obtained by robust decision trees and forests. On MNIST and Fashion-MNIST, provably robust boosting outperforms GROOT forests on robustness by respectively 0.8% and 5.5% but takes 122 times and 162 times longer to train. We implement and publish GROOT's source code in a Scikit-learn (Pedregosa et al., 2011) compatible classifier. We take inspiration from TREANT and allow users to easily configure the perturbation range for each separate feature. Our main contributions are:

- An efficient score function that allows us to fit trees 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. Related Work

To the best of our knowledge we summarize all related works on robust decision tree learning. In Table 1 we com-

[1]Cyber Security Group, Delft University of Technology, Delft, The Netherlands. Correspondence to: Daniël Vos <d.a.vos@tudelft.nl>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

Table 1: Overview of algorithms for fitting robust decision trees. Runtimes in terms of n number of samples, all algorithms also grow linearly in number of features and exponentially in depth.

Algorithm	Runtime	Threat model
GROOT	$\mathcal{O}(n \log n)$	L_∞ and variations
Provably robust boosting	$\mathcal{O}(n^2)$	L_∞
TREANT	$\mathcal{O}(n^2)$	axis-aligned rules
Chen et al. exact	$\mathcal{O}(n^2)$	L_∞
Chen et al. heuristic	$\mathcal{O}(n \log n)$	L_∞
MILP hardening	$\mathcal{O}(n \log n)$	L_0 / L_1 / L_2 / L_∞
Approx. hardening	$\mathcal{O}(n \log n)$	L_0

pare each algorithm's runtime complexity and threat model. We assume the reader to be familiar with regular decision tree learning algorithms.

2.1. Hardening Tree Ensembles

Setting the foundations of robust decision trees, Kantchelian et al. (Kantchelian et al., 2016) propose a hardening approach for tree ensembles and prove that finding adversarial examples under distance constraints is NP-hard for tree ensembles. They also provide a MILP formulation to solve the problem for arbitrary L_p norm which we use to verify the robustness of our models in Section 6.

2.2. Robust Decision Trees

Chen et al. (Chen et al., 2019) present an algorithm that fits robust decisions trees against L_∞ norm perturbations by using a new splitting criterion. This criterion is the worst case information gain or Gini impurity when an attacker moves points within an L_∞ radius. The authors find that they can compute the criterion exactly using gradient descent which takes $\mathcal{O}(n)$ time in terms of n samples. They deem this computation intractable for boosting ensembles and therefore give a fast heuristic based on four representative cases. GROOT's criterion is equivalent to the exact criterion but speeds up the computation to $\mathcal{O}(1)$ time and thereby enables its use in ensembles.

2.3. 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 still greedily builds a tree but it directly optimizes a loss function instead of using a splitting criterion. Although this allows TREANT to train against a variety of attackers, their algorithm deploys a solver to optimize the loss function and pre-computes all possible attacks which takes in the order of hours to run.

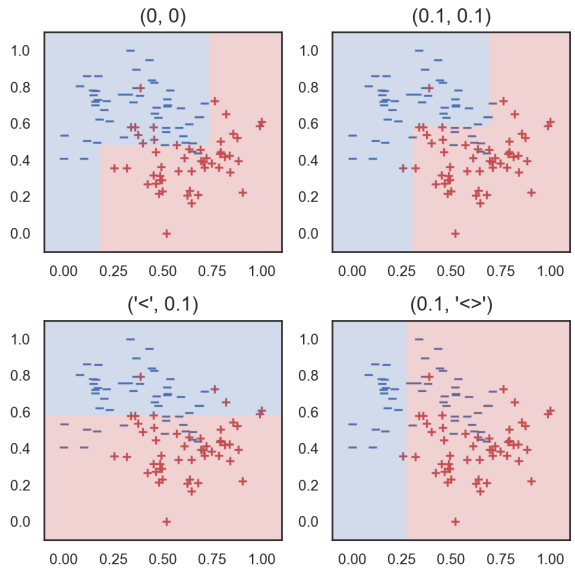


Figure 1: Decision regions of GROOT trees attacked by different threat models (indicated above each image). The threat model greatly influences the learned trees, e.g. robust decision trees against L_∞ perturbations (top right) are different from trees robust against other attackers (bottom).

2.4. Provably Robust Boosting

Where Chen et al. and TREANT describe algorithms for fitting a single robust tree, provably robust boosting (Andriushchenko & Hein, 2019) directly fits a robust ensemble. The authors find that they can efficiently compute the adversarial loss for boosted decision stumps and use this to derive an upper bound on the adversarial loss of boosted decision trees. By optimizing this bound on boosted decision trees they reach state-of-the-art performance on adversarial accuracy in tree ensembles and can compete with results from neural networks. While their ensembles contain many shallow trees which grants fast inference time, the training time of the method is in the order of hours.

3. Specifying Threat Models

In our work we assume existence of an attacker that knows the model 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 only be perturbed to a higher value.
- $<$: This feature can only be perturbed to a lower value.
- $<>$: This feature can be perturbed to any value.
- ϵ: The feature can be perturbed by a distance of ϵ.
- (ϵ_l, ϵ_r): The feature can be perturbed ϵ_l left or ϵ_r right.

We visualize GROOT trees with a variety of threat models in Figure 1. 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 ϵ as (ϵ, ϵ). For conciseness we only use the tuple notation in the algorithms in section 5. When we set the threat model to ϵ for each feature, it behaves identically to an L_∞ norm. We also allow the user to choose whether one or both of the classes can be perturbed.

In the rest of the paper we use an L_∞ threat model where both classes move and where features are scaled to the range $[0, 1]$ since this allows us to compare to existing work. To foster further research we implemented GROOT according to the Scikit-learn API and published the code on GitHub[1].

4. Adversarial Gini Impurity

Similar to robust decision trees (Chen et al., 2019) we use the worst-case Gini impurity to score threshold values. We show that we can efficiently compute this function by leveraging its concavity.

4.1. Adversarial Gini Impurity for Two Moving Classes

We typically fit decision trees with a splitting criterion such as the Gini impurity. To determine the quality of a split we then take the weighted average of the scores on both sides. We can define the Gini impurity for two classes as:

$$G(n_0, n_1) = 1 - \left(\frac{n_0}{n_0+n_1}\right)^2 - \left(\frac{n_1}{n_0+n_1}\right)^2 \tag{1}$$

Where n_0 and n_1 are the number of samples of label 0 and 1 respectively. Then we combine this into a score function by taking the weighted average with respect to number of samples on each side of the split (other works use the Gini gain which behaves identically):

$$S(l_0, l_1, r_0, r_1) = \frac{(l_0 + l_1) \cdot G(l_0, l_1) + (r_0 + r_1) \cdot G(r_0, r_1)}{l_0 + l_1 + r_0 + r_1} \tag{2}$$

Where l_0 and l_1 are the number of samples on the left side of the split of label 0 and 1 respectively. Similarly r_0 and r_1 represent samples on the right. Normally one searches for a split that minimizes this score function. Instead, we keep track of a set I that contains all samples close enough to the split to cross it under adversarial influence. We minimize the score function after the attacker maximizes it by perturbing the samples in I.

Where one normally minimizes the Gini impurity, we assume an attacker that aims to maximize $S(l_0, l_1, r_0, r_1)$ by moving samples from I to different sides of the split. We visualize this maximization problem in Figure 2. Here, i_1 is the number of points with label 1 that are close enough to the split that the adversary can move them to either side. Mathematically we are looking for the integer $m_1 \in [0, i_1]$ such that m_1 points of I_1 move to the left side of the split and $i_1 - m_1$ to the right. Similarly we have an i_0 and m_0 for the class 0 samples. The score function under attacker influence is:

$$S_{\text{robust}}(l_0, l_1, r_0, r_1, i_0, i_1) = \max_{m_1 \in [0,i_1], m_0 \in [0,i_0]} S(l_0 + m_0, l_1 + m_1, r_0 + i_0 - m_0, r_1 + i_1 - m_1) \tag{3}$$

We can then write the m_1' and m_0' that maximize it as:

$$m_1', m_0' = \underset{m_1 \in [0,i_1], m_0 \in [0,i_0]}{\arg\max} \left(\frac{(l_0 + m_0)(l_1 + m_1)}{l_0 + l_1 + m_1 + m_0} + \frac{(r_0 + i_0 - m_0)(r_1 + i_1 - m_1)}{r_0 + r_1 + i_0 + i_1 - m_1 - m_0} \right) \tag{4}$$

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

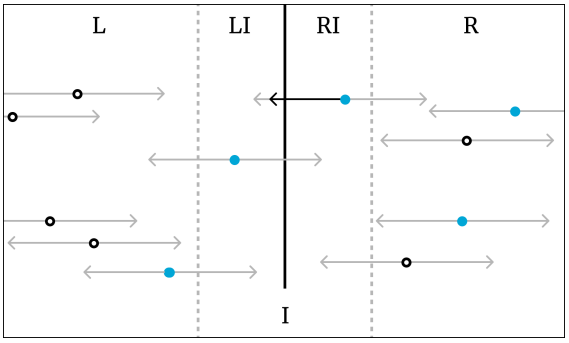


Figure 2: Example of the adversarial Gini impurity where samples can move in the range of the arrows. We want to move a number of samples from I over the threshold (line, center) to maximize the weighted average of Gini impurities. In this example we can move the single blue (filled) sample from RI into LI to maximize it.

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

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):

$$m'_0 = \frac{l_1(r_0 + i_0) - l_0(r_1 + i_1)}{l_1 + r_1 + i_1} + \frac{(l_0 + r_0 + i_0)m'_1}{l_1 + r_1 + i_1} \quad (5)$$

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 m_1 and m_0. 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 of m_1, m_0 ($|LI_1|, |LI_0|$) then round to the nearest integers.

Computing the point and rounding it takes $\mathcal{O}(1)$ time. Therefore we have an efficient method (constant time) to compute S_{robust} (Equation 2).

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 $\mathcal{O}(n \log n)$ time in terms of n samples. Similar to these algorithms, GROOT greedily makes splits according to a heuristic. This strategy performs well in practice but has no provable bound (Kearns, 1996).

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 Section 4 to find the split that is accurate against an adversary. Below we describe our algorithm for numerical features, for categorical features we refer to the Appendix. The time complexity in terms of n samples for both cases is bounded by $\mathcal{O}(n \log n)$ per feature.

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 0.4 that can be perturbed in a radius of 0.1, we score a split at 0.3, 0.4 and 0.5. At the start of Algorithm 1 we sort all candidate splits. We can then compute the sample counts and evaluate each split in $\mathcal{O}(1)$ time, as explained in Section 4. Recall that we consider at maximum $3n$ splits, where n is number of samples. Therefore the time complexity of evaluating splits is $\mathcal{O}(n)$ per feature and this means the fitting run time is dominated by the sorts of complexity $\mathcal{O}(n \log n)$.

Algorithm 1 Find Best Robust Split on Numerical feature

Input: feature values X, perturbation limits (ϵ_l, ϵ_r)
X_1 refers to the samples with label 1

$S \leftarrow X \cup \{o - \epsilon_l | o \in X\} \cup \{o + \epsilon_r | o \in X\}$
for $s \in S$ **do**
  $R \leftarrow \{o | o \in X \wedge o > s + \epsilon_r\}$
  $RI \leftarrow \{o | o \in X \wedge s < o \leq s + \epsilon_r\}$
  $LI \leftarrow \{o | o \in X \wedge s - \epsilon_l < o \leq s\}$
  $L \leftarrow \{o | o \in X \wedge o \leq s - \epsilon_l\}$
  $(m_{1s}, m_{0s}) \leftarrow$ number of samples from I to move left {See Eq. 5}
  $m_{1s} \leftarrow \text{round}(m_{1s}), \quad m_{0s} \leftarrow \text{round}(m_{0s})$
  $g_s \leftarrow S(|L_0| + m_{0s},\ |L_1| + m_{1s},\ |R_0| + |I_0| - m_{0s},\ |R_1| + |I_1| - m_{1s})$ {See Equation 2}
end for
$s' \leftarrow \arg\min_s\ g_s$
split with threshold s'

Output: $(s', g_{s'}, m_{1s'}, y_{s'})$

Algorithm 2 Fit Robust Tree on Numerical Data

Input: sample set X, perturbation limits (ϵ_l, ϵ_r)
X_1 refers to the samples with label 1

if stopping criterion (e.g. maximum depth) **then**
  create Leaf($|X_0|, |X_1|$)
else
  for $f \leftarrow 1...F$ **do**
    $s_f, g_f, m_{1f}, m_{0f} \leftarrow \text{BestRobustSplit}(X^f, \epsilon_l, \epsilon_r)$
  end for
  $f' \leftarrow \arg\min_f\ g_f$
  determine R, RI, LI, L for split f' as in Alg. 1
  move $m_{1f'}$ random samples from I_1 to LI_1, the remaining $|I_1| - m_{1f'}$ samples go to RI_1
  move $m_{0f'}$ random samples from I_0 to LI_0, the remaining $|I_0| - m_{0f'}$ samples go to RI_0
  $\text{node}_l \leftarrow \text{FitRobustTree}(L \cup LI)$
  $\text{node}_r \leftarrow \text{FitRobustTree}(R \cup RI)$
  create DecisionNode($s_{f'}, \text{node}_l, \text{node}_r$)
end if

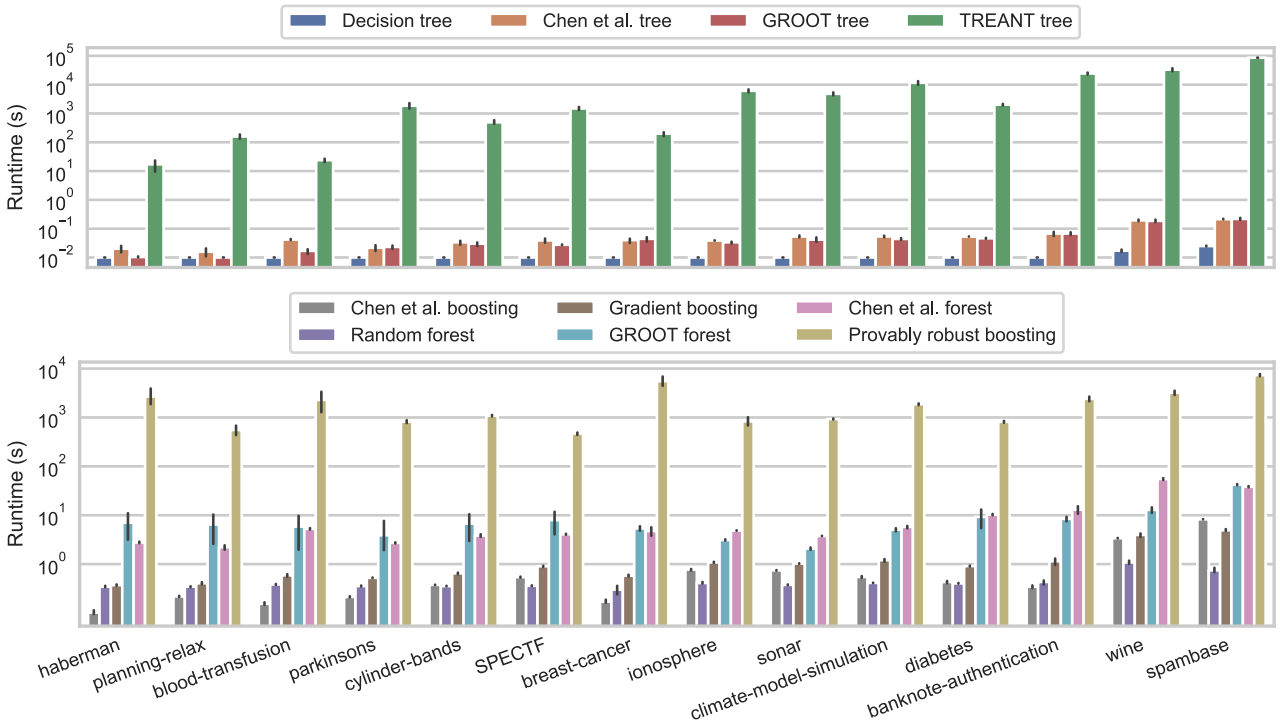


Figure 3: Runtimes of decision trees and ensembles in seconds on a logaritmic scale. Decision trees, random forests and gradient boosting enjoy Scikit-learn's optimized implementation. GROOT and Chen et al. consistently run 100-1000 times faster than TREANT and provably robust boosting. TREANT's spambase runs were terminated after 24 hours.

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. In our robust trees we account for samples that the adversary moves by modifying this propagation which we define in Algorithm 2. We do not only keep track of left (L) and right (R) samples, but also store an 'intersection' set $I = LI \cup RI$ that contains samples that can move to both sides. Here LI are the samples from I that were originally on the left side and RI were originally on the right side.

In section 4 we showed that m'_1 and m'_0 are the optimal values for the adversarial Gini impurity. From I_1 we move samples over the split to place m'_1 samples on the left and $I_1 - m'_1$ on the right. If there were, before moving, fewer than m'_1 samples on the left we move samples from the right. If there were more samples on the left we move them to the right. We do this to keep as many samples as possible on the original side of the split. We repeat the same procedure for m_0 and I_0.

The actual samples that move are randomly selected from the intersection I this makes our algorithm non-deterministic and therefore with different randomization seeds the algorithm can fit different trees. Our intuition behind random selection is that it prevents influence on the data distribution in splits further down the tree. If we were to move e.g. the closest samples to the threshold over the split, these samples might correlate with other features and cause any side of the split to become biased to specific values of that feature. The strategy differs between methods, e.g. TREANT chooses to move a sample to the side where it currently incurs the greatest loss. In future works, one could fit trees by optimizing all splits at once instead of a greedy method. In that case it is not needed to implement a sample propagation strategy but algorithms for optimal decision trees come at the cost of runtime.

5.3. GROOT Random Forests

To enable its use in ensembles we also implement a random forest of GROOT decision trees. In random forests it is important that the individual models have low covariance which we achieve using the same techniques as regular random forests (Breiman, 2001). Specifically, we train each decision tree on a bootstrap sample of the original training set and limit each decision node to scanning a random selection of $\sqrt{f}$ features (given f the total number of features). We do not limit the size of the decision trees. Similarly to Scikit-learn's implementation of random forests, the ensemble makes predictions by averaging and then rounding all individual tree predictions.

6. Results

We present results on 14 structured datasets as well as MNIST (LeCun et al., 2010) and Fashion-MNIST (Xiao et al., 2017). We compare GROOT against regular tree based models, the methods by Chen et al., TREANT and provably robust boosting. For the regular models we use scikit-learn's (Pedregosa et al., 2011) implementation as it is widely used for research in the field. The used hyper-parameters are summarized in Table 5. All datasets can be retrieved from OpenML[2], their specific versions, size and corresponding ϵ values can be found in Table 2. We removed any data row with missing values as it is unclear how to measure robustness against these samples.

6.1. Training Runtime

To compare the efficiency of the algorithms, we plot the run times of each run in Figure 3, the results shown are averaged over 5 data folds. 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 and therefore did not use any parallel optimizations.

Regarding the single decision tree models, regular decision trees enjoy the optimized code by Scikit-learn which is clearly the fastest. Comparing TREANT and GROOT, we see that our algorithm runs three to six 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 only uses a different splitting criterion, which we implemented in the code of GROOT.

In the ensemble model results we again see very fast results from the optimized implementations by Scikit-learn (Random forest and Gradient boosting) and Chen et al. boosting which is built on XGBoost (Chen & Guestrin, 2016). Still, GROOT and the Chen et al. forest run 2 to 3 orders of magnitude faster than provably robust boosting.

6.2. Predictive Performance on Structured Data

To determine the quality of the models produced by each algorithm we measure adversarial accuracy using the exact MILP attack (Kantchelian et al., 2016) which we modified to a feasibility problem as done in (Andriushchenko & Hein, 2019) to improve run time. The adversarial accuracy is the accuracy after samples have been optimally perturbed within an L_∞ ball of radius ϵ.

We encoded the above threat models in TREANT's attack rules using precondition $[-\infty, \infty]$ and postcondition ϵ or $-\epsilon$. Each rule has cost 1 and the attacker has a budget equal to the depth of the trees. In the original TREANT implementation, it allows attack rules to be applied to the same feature multiple times (in each decision node), resulting in attacks larger than ϵ. 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_∞ radius attack model.

Another popular method for measuring robustness (e.g. in (Kantchelian et al., 2016; 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 each model on each dataset with 5 fold stratified cross validation. All single decision trees 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 before-mentioned minor modifications. Scikit-learn uses the Gini impurity and TREANT optimizes the sum of squared errors. All models required at least 10 samples to make a split and 5 samples to create a leaf. We allowed all models to split multiple times on the same feature. All ensembles were limited to training 100 trees. We report the average adversarial accuracy and regular accuracy over 13 of the 14 structured datasets in Figure 4. We present the number of wins and average ranks over the datasets in Table 3. In both, we left out the results on spambase as TREANT did not finish fitting after multiple days of running.

While all models considered scored well on accuracy, the scores in the adversarial setting differ. Regarding single

Table 2: Structured datasets used by OpenML (version) name. Whenever possible, ϵ is taken from earlier work.

Dataset	Samples	Features	ϵ
banknote-authentication (1)	1372	4	0.1
blood-transfusion (1)	748	4	0.1
breast-cancer (1)	683	9	0.3
climate-model-simulation (4)	540	18	0.1
cylinder-bands (2)	277	37	0.1
diabetes (1)	768	8	0.05
haberman (1)	306	3	0.1
ionosphere (1)	351	34	0.2
parkinsons (1)	195	22	0.1
planning-relax (1)	182	12	0.1
sonar (1)	208	60	0.1
spambase (1)	4601	57	0.05
SPECTF (2)	267	44	0.1
wine (1)	6497	11	0.05

[2] https://www.openml.org/

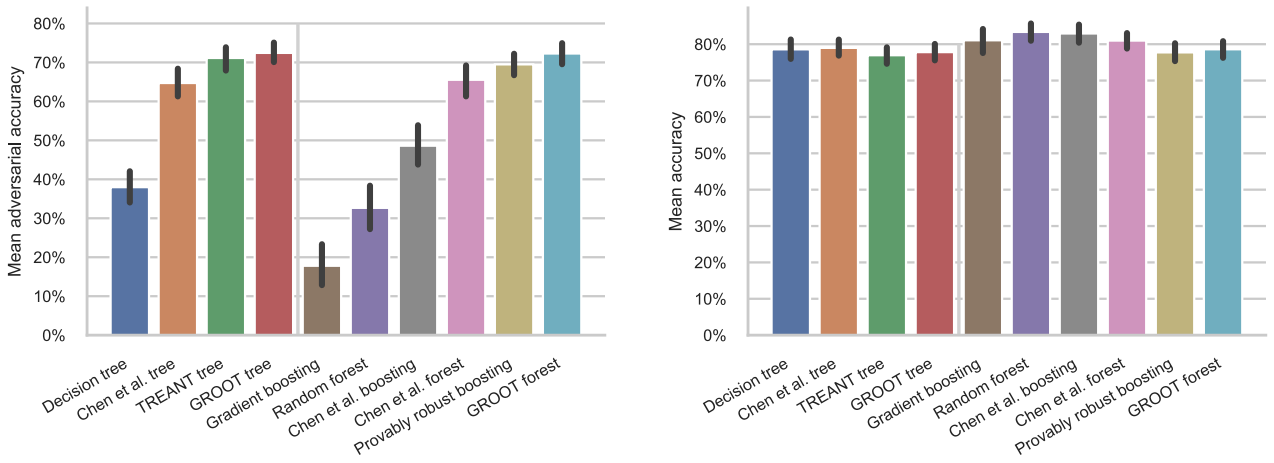


Figure 4: Average adversarial accuracy (left) and accuracy scores (right) over 13 structured datasets. TREANT and GROOT fit the most robust decision trees while provably robust boosting and GROOT random forests fit the most robust ensembles. More robust models tend to score up to 5% worse on regular accuracy.

trees, TREANT and GROOT perform similarly on adversarial accuracy and both significantly improve on regular decision trees by approximately 33%. GROOT obtains just over 1% more adversarial accuracy on average than TREANT, only one more win, but does obtain a higher mean overall rank. The heuristic by Chen et al. score approximately 7% worse than TREANT and GROOT. The individual results for each dataset are given in the appendix.

The ensemble results show that the GROOT random forest and provably robust boosting clearly achieve the best adversarial accuracy scores by about 5% difference over Chen et al. forest. Moreover, GROOT performs about 2.5% better than provably robust forest on average and obtains the best mean rank across all methods. The results from Chen et al. boosting were significantly lower than the scores from the random forest. We expect that the boosting model is more sensitive to the specific hyperparameters and that one could improve the scores using a hyperparameter search.

Interestingly, there is no clear difference between the best ensemble models and single decision tree models with regards to adversarial accuracy. This is in contradiction with the regular accuracy scores of decision trees and random forests. In those scores we see a clear 5% difference.

6.3. Predictive Performance on Images

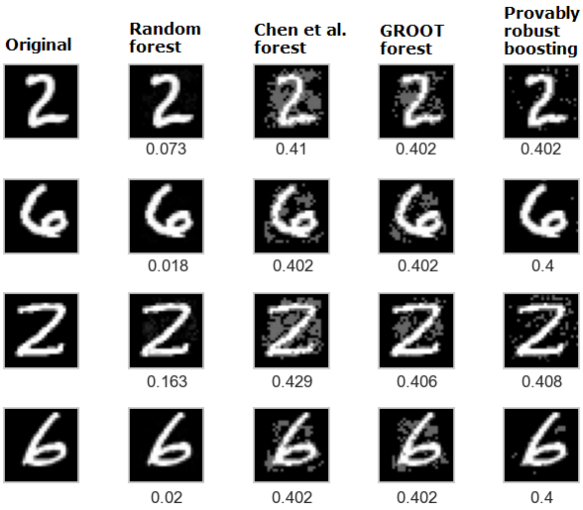
To compare predictive performance on image data we present results on MNIST and Fashion-MNIST. GROOT is limited to binary classification problems so we modify the datasets to MNIST 2 vs 6 and Fashion-MNIST sandals vs sneakers, similar to what previous works have done (Kantchelian et al., 2016; Chen et al., 2019; Andriushchenko & Hein, 2019). The prediction scores and runtimes are given in Table 4. On these datasets we ran provably robust boosting with parallelization enabled. All models ran on a system with 8GB RAM and 4 Intel i7-4710MQ CPU cores (8 logical cores), the hyperparameters were the same as in the previous experiment. The datasets were randomly split in a 70%-30% stratified train-test split and were evaluated against an L_∞ radius of 0.4.

Table 3: Summary of relative adversarial accuracy scores on 13 structured datasets.

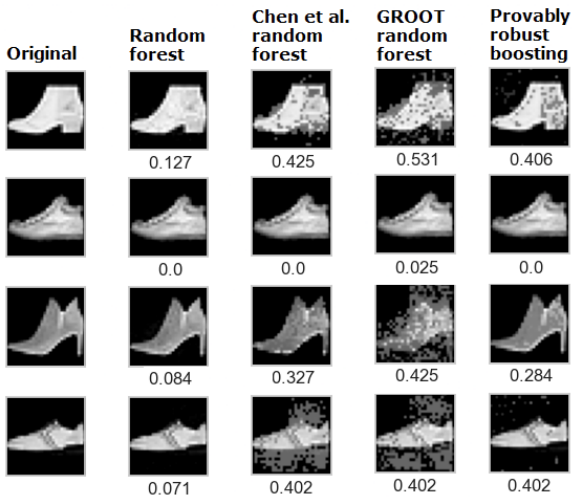
Model	Nr. Wins	Mean rank
Chen et al. boosting	0	7.5
Chen et al. forest	3	4.3
Chen et al. tree	0	5.1
Decision tree	0	7.9
GROOT forest	**9**	**1.5**
GROOT tree	7	1.8
Gradient boosting	0	9.8
Provably robust boosting	2	3.7
Random forest	0	8.2
TREANT tree	6	2.7

On both MNIST and Fashion-MNIST provably robust boosting achieved the best scores on adversarial accuracy. GROOT significantly improved on the adversarial accuracy scores of the other two models and scores close (0.8% difference) to the adversarial accuracy of provably robust boosting on MNIST. GROOT achieves these scores while running more than 100 times faster.

Previous works (Tsipras et al., 2018) have described the



(a) MNIST 2 vs 6



(b) Fashion-MNIST sandals vs sneakers

Figure 5: Minimal adversarial examples, L_∞ distances from the original are given below each example, larger is better. Some images already get misclassified without modification.

Table 4: Comparison of tree ensembles on image data. Provably robust boosting achieves the best adversarial accuracy but takes in the order of hours to run while GROOT runs in minutes and significantly improves on other fast models.

MNIST 2 vs 6			
Model	Acc.	Adv. acc.	Time
Random forest	99.7%	0.0%	3.9 sec.
Chen et al. forest	98.9%	89.1%	2.1 min.
GROOT forest	99.4%	91.9%	2.5 min.
Provably robust boosting	99.2%	92.7%	5.1 hr.
Fashion-MNIST sandals vs sneakers			
Random forest	95.8%	0.0%	5.9 sec.
Chen et al. forest	90.0%	52.6%	4.3 min.
GROOT forest	89.0%	70.4%	5.0 min.
Provably robust boosting	88.9%	75.9%	13.5 hr.

accuracy-robustness trade-off and we find it here too. Particularly on the Fashion-MNIST dataset the most robust model sacrifices 6.9% regular accuracy where on the MNIST dataset this difference is 0.5%.

Figure 5 shows the minimal L_∞ norm perturbations required to change the prediction of each model. We generate the adversarial examples using a MILP formulation for attack tree ensembles (Kantchelian et al., 2016). The random forest models need visibly more perturbed feature values than the provably robust boosting model so by optimizing the L_∞ norm they also increase robustness in the L_0 norm. We expect this is due to the random forest approach of training each decision node on a limited selection of features which causes more variety in the selected features.

7. Discussion and Conclusions

We present GROOT, an algorithm for learning robust decision trees. It uses an analytical solution for computing the adversarial Gini impurity and by doing so runs two to six orders of magnitude faster than the state-of-the-art approaches. Our results show that GROOT trees score competitively with TREANT and so do GROOT random forests with provably robust boosting. While the single GROOT trees are the same size as those of TREANT, there is a noticeable difference between the size of random forest and boosting models. In the case of random forests we do not limit the size of the trees, where gradient boosting trees were trained to a maximum depth of 8 (this is intended and does not reduce model performance). This means that while our method trains quickly, the models suffer from relatively longer inference and robustness verification times. In further research, the random forests may be post-processed to reduce the size.

For the sake of comparison, we experimented on the same public datasets that similar works on robustness used, but these datasets were originally not intended for research into adversarial attacks. In the near future, we will apply the methods we discussed to problems where adversarial modifications are an important concern such as fraud, malware and intrusion detection. Chen et al. (Chen et al., 2021) have successfully applied robust decision trees for such security applications. Our fast splitting criterion can also be used to speed up their algorithm.

Table 5: Hyperparameters of all models used in our experiments. Parameters that were not applicable were left blank. Except for n_estimators, the values were copied from their original works wherever possible.

Parameter	Decision tree	Chen et al. tree	GROOT tree	TREANT tree	Random forest	Gradient boosting	Chen et al. boosting	Chen et al. forest	GROOT forest	Provably robust boosting
max_depth	4	4	4	4	None	8	8	None	None	8
min_samples_split	10	10	10	10	10	10	-	10	10	10
min_samples_leaf	5	5	5	-	5	5	-	5	5	5
n_estimators	-	-	-	-	100	100	100	100	100	100
η	-	-	-	-	-	-	0.2	-	-	0.2
γ	-	-	-	-	-	-	1.0	-	-	-
min_child_weight	-	-	-	-	-	-	1	-	-	-
affine	-	-	-	False	-	-	-	-	-	-

While greedy algorithms that choose locally optimal splits are popular for fitting decision trees, they can theoretically perform arbitrarily poorly. There have been many successful efforts in training optimal decision trees (Bertsimas & Dunn, 2017; Rhuggenaath et al., 2018; Verwer & Zhang, 2019; Aglin et al., 2020) and their results show that the greedy algorithms come close to optimal performance. An optimal algorithm for robust decision trees will determine whether the greedy approaches for robust trees perform as well as their regular counterparts.

We conclude that:

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

References

Aglin, G., Nijssen, S., and Schaus, P. Learning optimal decision trees using caching branch-and-bound search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3146–3153, 2020.

Andriushchenko, M. and Hein, M. Provably robust boosted decision stumps and trees against adversarial attacks. *arXiv preprint arXiv:1906.03526*, 2019.

Bertsimas, D. and Dunn, J. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, 2017.

Breiman, L. Random forests. *Machine learning*, 45(1): 5–32, 2001.

Calzavara, S., Lucchese, C., Tolomei, G., Abebe, S. A., and Orlando, S. Treant: Training evasion-aware decision trees. *Data Mining and Knowledge Discovery*, pp. 1–31, 2020.

Chen, H., Zhang, H., Boning, D., and Hsieh, C.-J. Robust decision trees against adversarial examples. In *ICML*, pp. 1122–1131, 2019.

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

Chen, Y., Wang, S., Jiang, W., Cidon, A., and Jana, S. Cost-aware robust tree ensembles for security applications. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., and Madry, A. Adversarial examples are not bugs, they are features. In *Advances in Neural Information Processing Systems*, pp. 125–136, 2019.

Kantchelian, A., Tygar, J. D., and Joseph, A. Evasion and hardening of tree ensemble classifiers. In *ICML*, pp. 2387–2396, 2016.

Kearns, M. Boosting theory towards practice: Recent developments in decision tree induction and the weak learning framework. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 1337–1339, 1996.

LeCun, Y., Cortes, C., and Burges, C. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Meng, D. and Chen, H. Magnet: a two-pronged defense against adversarial examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 135–147, 2017.

Papernot, N., McDaniel, P., and Goodfellow, I. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016a.

Papernot, N., McDaniel, P., Wu, X., Jha, S., and Swami, A. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 582–597. IEEE, 2016b.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Rhuggenaath, J., Zhang, Y., Akcay, A., Kaymak, U., and Verwer, S. Learning fuzzy decision trees using integer programming. In *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pp. 1–8. IEEE, 2018.

Samangouei, P., Kabkab, M., and Chellappa, R. Defense-GAN: Protecting classifiers against adversarial attacks using generative models. In *International Conference on Learning Representations*, 2018.

Szegedy, C., Zaremba, W., Sutskever, I., Estrach, J. B., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014*, 2014.

Tsipras, D., Santurkar, S., Engstrom, L., Turner, A., and Madry, A. Robustness may be at odds with accuracy. *arXiv preprint arXiv:1805.12152*, 2018.

Verwer, S. and Zhang, Y. Learning optimal classification trees using a binary linear program formulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 1625–1632, 2019.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.