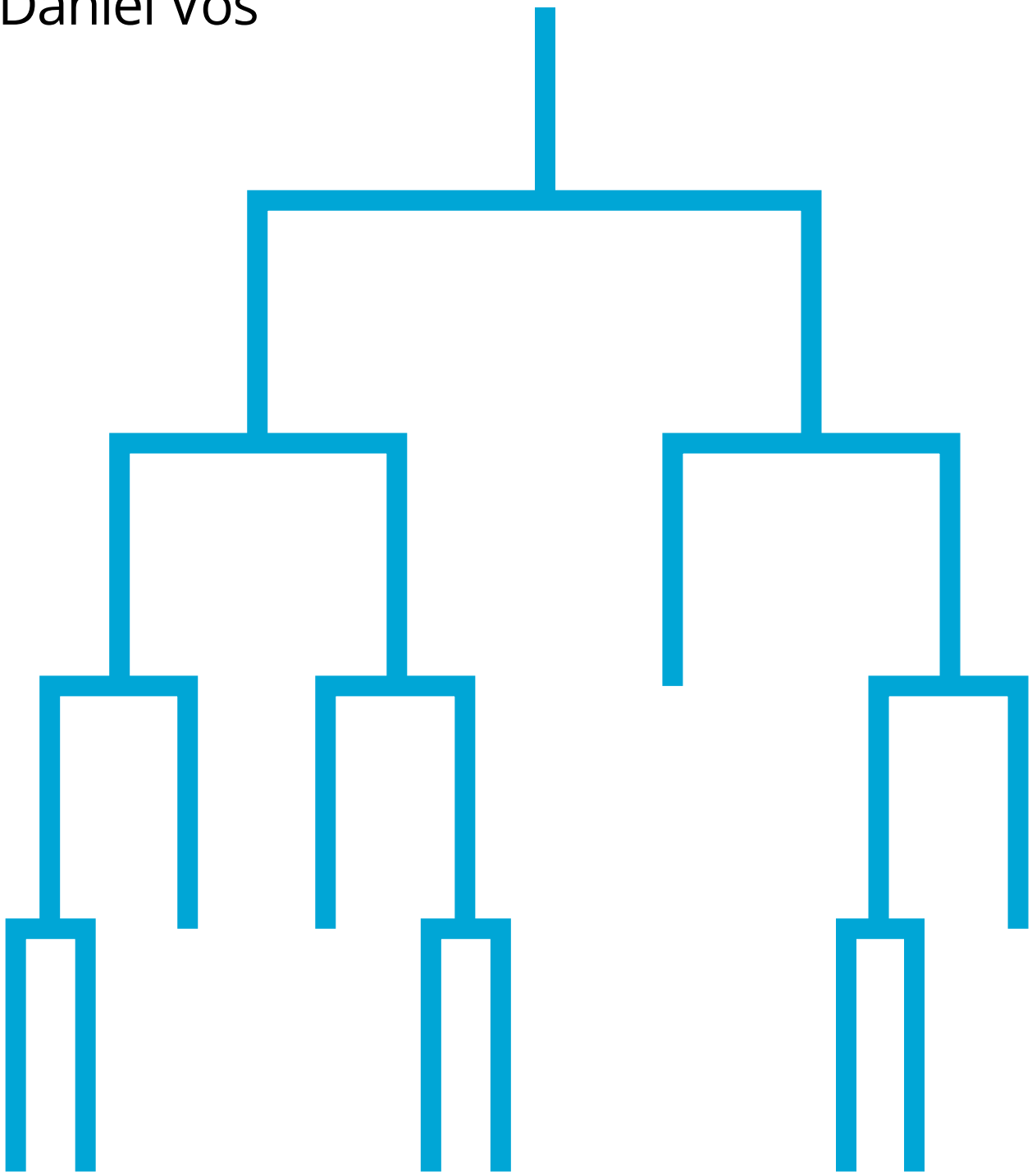
Adversarially Robust Decision Trees Against User-Specified Threat Models

Daniël Vos



TU Delft

Master thesis

July 2020

Adversarially Robust Decision Trees Against User-Specified Threat Models

by

Daniël Vos

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday July 1, 2020 at 9:00 AM.

Student number:	4446704	
Project duration:	November 11, 2019 – July 1, 2020	
Thesis committee:	Prof. dr. ir. R. l. Lagendijk,	TU Delft
	Dr. S. Verwer,	TU Delft, supervisor
	Dr. M. Loog,	TU Delft

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

TU Delft

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 that is two orders of magnitudes faster and scores 4.3% better on accuracy against adversaries moving all samples than the state-of-the-art work while accepting 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 threat model specifying either a maximum distance or constraints on the direction of perturbation. Additionally we introduce two hyperparameters ρ and ϕ that can control the trade-off between accuracy vs robustness and accuracy vs fairness respectively. Using the hyperparameters we can train models with less than 5% difference in false positive rate between population groups while scoring on average 2.4% higher on accuracy against adversarial attacks. Lastly, we show that our decision trees perform similarly to more complex random forests of fair and robust decision trees.

Preface

You are reading the thesis "Adversarially Robust Decision Trees Against User-Specified Threat Models" which contains methods and experiments for fitting decision trees that are adversarially robust and fair. It was written to fulfill the requirements of the Computer Science master with a specialisation in Cyber Security at the Delft University of Technology.

During the entire project I was supervised by Sicco Verwer. I want to thank him and Christian Hammerschmidt for their guidance in the form of weekly meetings, in which they never got tired of my questions and ideas. The thesis students working in the cyber security group always made me excited to return to the sixth floor where we all worked tirelessly. Especially I want to thank my fellow student Cas Buijs for many in-depth discussions on adversarial examples and decision tree learning. I am certain that our research group could not work as well without the help of Sandra Wolff who seems to know and encourage every single person there. Lastly I want to thank my family and friends who kept me motivated whenever I encountered issues in my project.

I hope you enjoy reading this thesis.

Daniël Vos
Delft, June 2020

Contents

1 Introduction

At this moment machine learning and artificial intelligence are increasingly applied in sensitive fields such as cyber security, health care and finance [6, 12]. Machine learning methods have been very successful due to their predictive power, but these methods have shortcomings with regards to interpretability, adversarial robustness and fairness that one should address before applying them.

Neural networks for example, are complicated models that can contain billions of learned parameters. While these models achieve state-of-the-art accuracy on tasks in computer vision [25] and natural language processing [33], their complex structures with many parameters are extremely hard for humans to understand. This could be acceptable for applications with low impact, but when we use machine learning for decisions that impact people's lives non-interpretability is a risk. One can wonder whether a neural network has made a decision due to strong evidence or as a random guess. For example in the case of credit scoring we need to know whether a model's behavior makes sense and does not depend on features that we want to avoid.

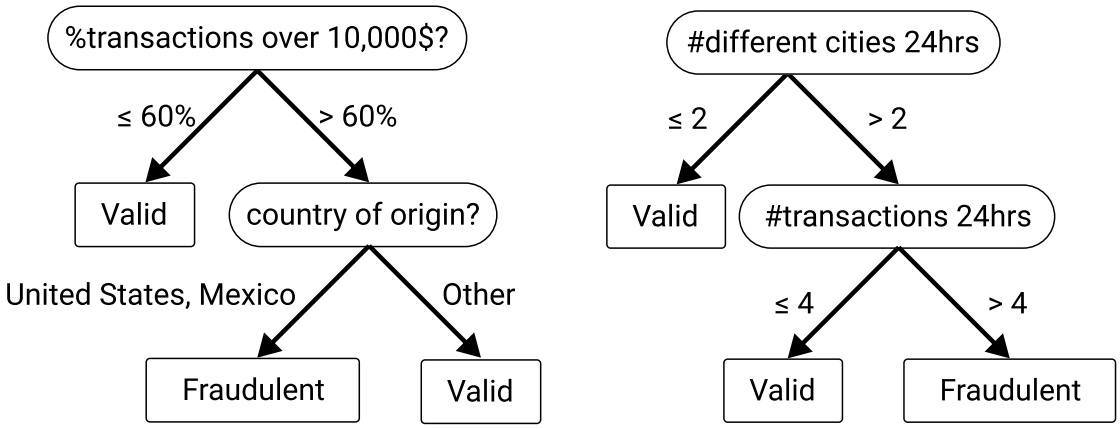
Recent research has discovered the existence of adversarial examples: inputs intentionally affected by small changes such that they are wrongly predicted [36]. Adversarial examples create a risk for when machine learning models are applied in sensitive areas. For example in security, where we often apply machine learning techniques for discriminating benign and malicious behavior. In such an application, malicious users can change their behavior to evade the model and be misclassified as benign. Say we have a program that detects email spam by checking whether the email contains many capital letters and exclamation marks. When spam creators learn about this detector they can easily change their emails to avoid detection.

Machine learning techniques can also learn unfair models that discriminate based on e.g. race, sex, age or other features. When we apply machine learning for decisions that impact people's lives, it is unacceptable that models contain such a bias. It was found that COMPAS, a tool that was used to determine whether people with criminal charges would re-offend, had a strong bias[1]. African American people were almost twice as likely to be incorrectly classified as high risk than Caucasian people.

To address the issue of non-interpretability in machine learning, research has previously used models such as decision trees. These models consist of a series of simple decision rules such as 'cost $\leq$ 30\$' that eventually lead to a prediction value such as 'benign'. By limiting the size of these models they are interpretable for humans. Although there is a large body of literature concerning decision tree learning there is limited work on training decision trees for adversarial robustness and fairness.

These methods for fitting adversarially robust decision trees are limited to ensembles, specific threat models or are limited in run time. Kantchelian et al. [22] propose the first hardening (adversarial training) approach for decision tree ensembles using a mixed-integer linear programming approach. They assume that the adversary is limited to perturbing inputs within a radius of arbitrary L norms (e.g. L_1, L_∞) and also give an approximation algorithm that supports the L_0 norm. When training against these L_0 attacks they show an increase in ensemble robustness under L_0, but worse performance under attacks with different norms.

[1] `https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing`



(a) Regular decision tree

(b) Robust and fair decision tree

Figure 1.1: Two decision trees for a hypothetical fraud detection task. Regular decision trees (a) are not trained to be robust against adversarial examples or to be fair towards different population groups. Adversaries can easily change their transaction amount and decisions based on country of origin can cause discrimination. Our robust and fair decision tree learning algorithm (b) attempts to fit decision trees that enjoy both of these properties by avoiding discriminating features and choosing features that are harder to manipulate.

Chen et al. [10] assume an attacker that is restricted by an L_∞ norm and give an algorithm to fit a decision tree that is more robust to these attacks. This algorithm closely follows the greedy tree building approach used to train regular decision trees. They also propose a gradient boosting decision trees (GBDT) approach that deploys a faster heuristic to train robust decision trees.

Calzavara et al. propose a more flexible robust learning procedure called TREANT [7], in which the user can describe an adversary using axis-aligned rules and attack budgets. TREANT then fits a decision tree by directly optimizing a loss function under attacker influence. Additionally their method provides 'attack invariance' which ensures their trees do not become less robust when grown larger.

As for robust decision trees; there also exist promising solutions for training fair decision trees. Kamiran et al. [21] have proposed and tested different ways of altering the decision tree learning algorithm to train fair and accurate trees efficiently. Recently, Aghaei et al. [1] used a Mixed Integer Linear Programming (MILP) approach to train fair trees slowly but with higher predictive accuracy. Although these works are promising, we have yet to see an algorithm for training decision trees both fairly and robustly. Our intuition is that by choosing splitting features that are less susceptible to adversarial attacks and less correlated with discriminatory features we can fit robust and fair trees. We will motivate this intuition with an example.

1.1. Motivating Example

Let us give an example of an application for which we desire interpretability, adversarial robustness and fairness. Consider a payment service provider in the Netherlands. According to the European Union regulations[2] this payment service provider would have to put effort into detecting transaction fraud. As fraud decisions can greatly impact people's lives we also want to be fair (not discriminate). One way to approach this would be by collecting a history of transactions with data about currency, identity, location and whether fraud occurred. The provider can then use this history to train a machine learning model that detects fraud. Given that we wanted to train an interpretable model we can use for example a linear classifier or decision tree but these models are not necessarily robust or fair.

Say a regular decision tree is used, the model might for instance learn that users who often make large transactions are likely to be fraudulent. However when an adversary discovers that decision rule, they can easily evade it by decreasing the monetary value of their transactions. To prevent this it would be desirable that one could communicate to the model that an adversary can influence the monetary value easily. In that case the model can potentially choose a different feature to base the decision on to prevent evasion. However, if the decision tree selects a more robust feature such as 'country of origin' or a similar feature that correlates

[2] https://www.toezicht.dnb.nl/3/50-237177.jsp

with it, there can be fairness issues. For instance, people from some countries might be falsely accused of fraud at a much greater rate than other countries. This same problem can occur for features related to sex, age or other identifying properties. We visualize such a decision tree that is neither (necessarily) robust or fair in Figure 1.1a.

Instead of training a decision tree with a typical learning algorithm, we want to communicate to the algorithm what features can be affected by an adversary. This way the learning algorithm can generate a tree that makes use of our input features in a more robust way. Additionally we want a model that maintains similar rates of wrong predictions among different population groups to prevent discrimination. We give an example of such a decision tree in Figure 1.1b.

1.2. Research objectives

In this work we attempt to create models that are interpretable, robust and fair. To achieve this we train decision trees of which we improve the robustness and fairness while trading off accuracy. The main question that we attempt to answer is therefore

"How can we train decision trees such that they are robust against user-specified adversarial attacks and fair for different population groups"

We break this question down into three parts: improving robustness, improving fairness and tree ensembles. In the first part we want to research efficient methods for training adversarially robust decision trees. Additionally we want the methods to work for attackers that users can specify so without assuming a single L_x norm attacker. Then we want to improve the fairness of decision trees while maintaining high robustness. Of specific interest is how to combine the methods for fair decision trees with methods for robust decision tree learning. Lastly, previous works have often resorted to ensemble methods [7, 10, 22] but this abandons interpretability. We wish to quantify the improvement in using tree ensembles over single decision trees to explore whether the impact on interpretability is worth it.

To summarize, we attempt to answer the following sub-questions:

1. How can we train robust decision trees against user-specified threat models efficiently?
2. How can we train decision trees robustly and simultaneously fairly?
3. To what extent do decision tree ensembles improve over single decision trees?

1.3. Contributions

In this work, we propose a new algorithm for training robust decision trees. Like Chen et al. [10], we closely mimic the greedy recursive splitting strategy that traditional decision trees use, but we include a threat model in the algorithm such that each candidate split is scored by its worst-case performance under the specified attack. We revise the regular tree fitting algorithm by replacing the Gini impurity with an adversarial Gini impurity score function and propagate perturbed samples to child nodes. Since this function is concave with respect to the number of modified data points, we can score splits efficiently using an analytical solution. This makes our algorithm two orders of magnitude faster than existing approaches.

Moreover, it scores 4.3% better on average on adversarial accuracy on the same setting as TREANT while accepting an intuitive and permissible threat model. Where previous works [7, 10, 22] focus on either robustness or accuracy, we recognize that in practice a model will likely be attacked in just a fraction of cases. We introduce a parameter ρ that can trade-off accuracy for regular users and robustness against attackers without influencing algorithm run-time. Our main contributions are:

- A new efficient algorithm for learning decision trees, outperforming the state of the art on both adversarial accuracy and runtime.
- A framework that allows a user to intuitively specify the kinds of possible attacks in terms of axis aligned perturbations.
- A parameter that governs the trade-off between regular and adversarial accuracy, allowing the user to learn models depending on their risk aptitude.

We then extend this robust decision tree learning algorithm with a splitting criterion that encourages more fair splits. For this algorithm we include an additional hyperparameter to control the accuracy-fairness trade off. Using our implementation we are the first work to fit decision trees that can trade off accuracy, robustness and fairness. Additionally, we implement an efficient algorithm for computing the robustness of decision tree ensembles. Our contributions here are:

- An efficient algorithm for learning decision trees that are accurate, robust and fair
- A quantification of the performance difference between random forests and decision trees.

1.4. Outline

The rest of this thesis is structured as follows: In chapter 2 we give background information and evaluate previous works on decision tree learning, robustness and fairness. In chapter 3 we introduce an algorithm for training robust decision trees against malicious users and measure its performance. In chapter 4 we improve this algorithm with a score function that can fit robust trees against attackers moving all classes. Then in chapter 5 we extend the algorithm to also consider fairness and study it. In chapter 6 we research whether we can use the robust and fair decision trees in an ensemble to give up interpretability for increased predictive performance. We discuss and conclude in chapters 7 and 8.

2

Literature Review

Decision trees are classical models that have been widely researched since the 1960s and as the field of machine learning evolves decision trees evolve as well. In this chapter we aim to give a background on decision tree learning and to introduce the current state-of-the-art. We first summarize the foundations of decision tree learning and then evaluate previous works on robust and fair machine learning. Lastly we determine the research gaps that we aim to fill.

2.1. Decision Tree Learning

Recently in the field of machine learning there have been many advances around neural networks, but while these models achieve state-of-the-art accuracy they cannot be interpreted by humans. This property makes neural networks an undesirable choice of model when the task at hand concerns impactful decisions, as they must be explainable. In the tasks found in cyber security, for example, we often differentiate between 'benign' and 'malicious' network traffic / transactions. It is vital that one can explain and understand the decisions made by the model.

A well-researched and human understandable type of model is the decision tree. The models are interpretable when we limit the depth of these models (by depth 5 in this work) because that forces the model to have at most 2^{depth} different prediction regions. Decision trees are comprised of nodes that perform a simple rule on the input and leaves that contain the prediction value. An example of two decision trees was given in Figure 1.1. By following the path through decision nodes, each input will correspond to a leaf that contains the prediction. In this work we focus on two-class decision trees, so models that predict one of two classes (e.g. 'good'/'bad', 'benign'/'fraudulent'). There are many different flavors of decision trees that differ in the type of variables used (numerical vs categorical), the type of splitting rules and how many splits are made at each node. We consider nodes that always split in two and the trees can contain splitting rules on both numerical and categorical variables. That is, a node can split a numerical variable with a $\leq$ rule or split a categorical variable with a rule that specifies which categories go left and which go right.

2.1.1. Greedy Splitting

Although earlier mentions of decision trees exist (e.g. [2]), decision tree learning was popularized by two greedy algorithms that are different in whether they use the Gini impurity (CART [4]) or information gain (ID3 [32]) criterion. This work focuses on the Gini impurity as both criteria are very similar in their behavior [35], but the information gain criterion is slower to compute due to its logarithm operation. The general idea behind decision tree learning is finding a split that separates the different classes in the data well and recursively continuing this operation. To efficiently compute categorical splits we use methods described by Coppersmith et al. [14]. As they have proved, we can simply sort categories by their ratio of benign vs malicious samples and iterate through the sequence to find the optimal split.

2.1.2. Optimal Decision Trees

CART and ID3 are fast algorithms, but they do not generally fit optimal decision trees. Exact algorithms have been proposed such as the one by Bennet and Blue [3]. However, such algorithms scale poorly as in 1976

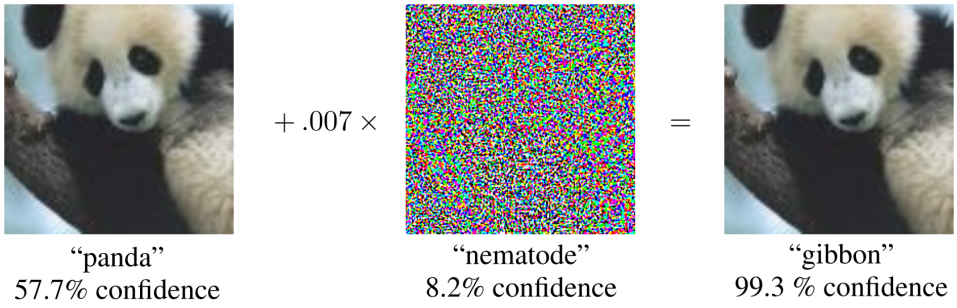


Figure 2.1: Classical illustration of adversarial examples from Goodfellow et al. [18]. GoogLeNet classifies the left image as a panda with 57.7% confidence but after adding specifically crafted noise (middle image), GoogLeNet misclassifies the resulting image as a gibbon with 99.3% confidence.

Hyafil and Rivest [24] showed that fitting optimal decision trees is NP-hard. Leveraging the constant improvements in optimizers such as Gurobi and CPLEX Verwer and Zhang [37] presented a Mixed-Integer Linear Programming (MILP) formulation for the entire decision tree learning problem. Such MILP formulations provide a felixble foundation for optimal decision tree learning as different objective functions can be easily encoded.

2.1.3. Tree Ensembles

Decision trees are often used in ensembles, groups of models that together function as one classifier. A common approach is random forests [5]. As ensembles tend to perform better when they are diverse, random forests create diversity by bootstrap aggregation (bagging) and by limiting the number of features each tree can consider for splitting.

Another ensemble method, and one of the most popular approaches in online machine learning competitions[1], is gradient boosting [16]. When applying gradient boosting to decision trees, trees are fitted one by one such that each new tree compensates for the weaker predictions of the trees before it. Particularly, implementations of gradient boosting decision trees such as XGBoost [13] have dominated in recent machine learning competitions.

It is worth noting that when using decision trees in (large) ensembles, the models are no longer explainable. This is because the averaging step of ensembles causes the number of distinct decision regions to grow fast when increasing ensemble size causing the models to be increasingly complex.

2.2. Adversarial Examples

During the initial development of machine learning algorithms, security was not considered. However, when impactful 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 also known as ‘adversarial examples’ [18], a visualization can be found in Figure 2.1. In this section we will discuss the first works on adversarial examples for neural networks and then present a taxonomy of the threat models that these works assumed.

2.2.1. Attacks

In 2014 Szegedy et al. [36] published a work on two intriguing properties of neural networks, one of them being the existence of adversarial examples. They showed that by adding only very small perturbations to the pixels of input images, they can cause almost any image to be misclassified. This paper sparked wide scale research into adversarial attacks and ways to defend against them. We first summarize four types of attacks against neural networks.

Fast Gradient Sign Method Where Szegedy et al. prove the existence of adversarial examples in neural networks using an optimization based approach, Goodfellow et al. [18] find a faster way to generate these examples. They introduce the ‘Fast Gradient Sign Method’, that by simply moving every pixel in the direction

[1] https://www.import.io/post/how-to-win-a-kaggle-competition/

of the gradient can reliably generate examples. The gradient can also be efficiently computed using one back-propagation operation. Using this fast method it becomes viable to train / test more robust neural networks.

Universal Perturbations Moosavi-Dezfooli et al. [27] show the existence of 'universal perturbations'. These perturbations, when added to input images, cause most input images to be misclassified regardless of the neural network used. The existence of universal perturbations suggest that adversarial examples might not be caused by the specific neural network but by the way neural networks extract information from the dataset.

Undetectable Perturbations Generally seen as the state-of-the-art approach to generating adversarial examples, Carlini and Wagner [9] propose a 100% effective optimization based approach against earlier defenses. At the time of publication there were already defenses proposed for hardening neural networks against adversarial examples such as defensive distillation (see next subsection). Carlini and Wagner's attack finds adversarial examples against this defense.

Transferability Papernot et al. [28] present the interesting finding that adversarial examples are transferable between models and even models of different kinds. They find that adversarial examples crafted for deep neural networks and linear regression models fool models of the same kind reliably, even when trained on a discrete subset of training data. Additionally, examples for neural networks, linear regression and k-nearest neighbors transfer well to SVMs. However, among the five model types tested, decision trees are the most susceptible to transfer attacks, motivating research into hardening decision trees.

2.2.2. Defenses

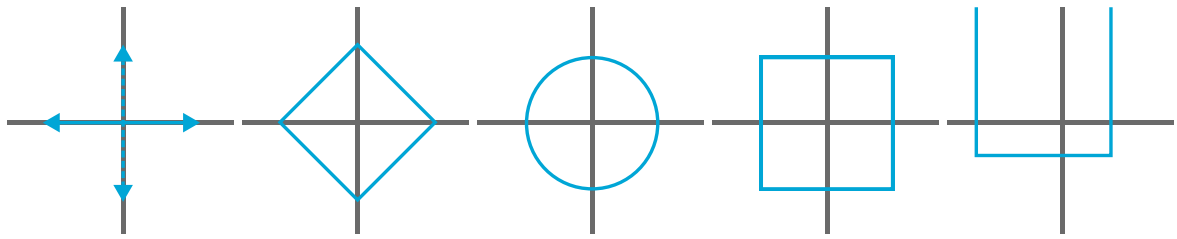
The discovery of adversarial examples has started an arms race in attacks and defences for neural networks. We discuss distillation as a defense, a detector and reformer based defense and one based on random forests.

Distillation The first practical defense to be proposed for neural networks is defensive distillation by Papernot et al. [29]. In their distillation procedure they first train a neural network in a typical way but add a second training step in which they train a new network on a relabeled dataset. They relabel each sample with the probability that the previously trained neural network assigns to the sample. This way one can smooth out the decision regions around samples to better protect against adversarial examples. Although their work proves to be more robust against attacks that generate adversarial examples by perturbing pixels sensitive to change, other works have found attacks that break the defense [9].

MagNet Meng and Chen [26] propose a two-step defense against the adversarial examples generated by the Carlini and Wagner attack [9]. By estimating the distribution of normal, unperturbed, inputs their defense builds detectors and reformers. The detectors aim to predict whether an input is an adversarial example and to reject it. If the sample is not rejected the reformer projects it back on the 'normal' samples manifold. The method highlights the importance of anomaly detection in preventing adversarial examples, as we cannot generally confidently predict inputs that are far from the training data.

Although Meng and Chen's defense seemed successful at first, Carlini and Wagner have responded with an article [8] on transfer attacks that break this defense. In the field of adversarial examples we see a back and forth competition between attacks and defenses. This demonstrates the weakness of neural networks in an adversarial setting and that this is not easily solved.

Defending Using Random Forests Ding et al. [15] propose an interesting defense for adversarial examples by combining neural networks and random forests. Their defense splits a trained neural network on a layer that amplifies perturbations and replaces the deeper part of the network by a random forest. They argue that because of the discontinuous nature of random forests adversarial examples will be harder to find. Their hybrid models perform only slightly worse than the full neural networks on accuracy while the models are much more robust against adversarial examples. This defense provides support for using tree-based models to improve accuracy against adversaries.



(a) L_0 (b) L_1 (c) L_2 (d) L_∞ (e) Axis-aligned

Figure 2.2: Perturbation limits used by different works. For Figure b-e the blue shapes specify the outlines for where a sample can move. The L_0 norm from Figure a limits how many features can be perturbed. In this case a sample can move everywhere over one axis, so the sample can be perturbed by either the filled arrow or dashed arrow. The axis aligned boxes from Figure e provide a more flexible way to describe perturbation limits, where perturbations can be asymmetrical or even unlimited.

2.2.3. Threat Models

With the many works on attacks against neural networks there is a variety in the assumptions that different researchers make. We attempt to give an overview of the different attackers that previous works have assumed and describe the attacker we assume.

Attacker capabilities The perturbations that are considered for adversarial examples (the threat model) are commonly limited by an L_x distance function. However, L_x norms capture only a small subset of the capabilities an attacker can have [17]. Therefore, this work extends the notion of possible attacks to also include asymmetric transformations on the axes and different transformations for different features. We give visualizations of L_x norms and the axis-aligned perturbation limits in Figure 2.2.

Attacker knowledge The threat model does not only describe what perturbations an attacker can likely produce, it also captures the knowledge we expect an attacker to have. Papernot et al. [30] present a model for attacker knowledge that can be summarized in three cases: white-box, gray-box and black-box. In a white-box case the attacker has full knowledge of the model under attack, including parameters and data. In a black-box setting attacks can often only use the model as a prediction oracle, but works use multiple variants (1 oracle call vs. k oracle calls, hard-label vs. soft-label). The gray-box setting is loosely defined to be in between white and black-box. An attacker might for example know the hyperparameters of the model but not the trained parameters.

Attacker goal Papernot et al. [30] also describe the attacker goal in their model. They describe four distinct goals: confidence reduction, misclassification, targeted misclassification and source-target misclassification. It is worth noting that except for confidence reduction, all these forms are equivalent for binary decision problems such as the ones we consider in this work.

Unlike some previous work, we do not assume that the adversary can move every data point. Instead in chapter 3 we assume that only malicious points will be moved by an adversary. This follows intuitively from the fact that in security, malicious traffic / transactions / occurrences are blocked or treated in a way that is bad for the user, so it is unlikely that a benign user will attempt to evade the model to incur negative consequences. In chapter 4 we change our attack model to one that moves both benign and malicious classes so we can compare to related works on similar adversaries.

Threat models in this work In this work we will always consider a white box threat model where an attacker aims to move malicious samples into benign decision regions. Therefore when we refer to user-specified threat models we mean the 'attacker capabilities'. We describe these attacker capabilities intuitively using the threat model given in section 3.1.

2.3. Robust Decision Trees

In this section we discuss works that our own algorithms are mostly built upon. We first discuss algorithms for decision tree learning and then briefly methods for testing the robustness of decision trees and tree ensembles.

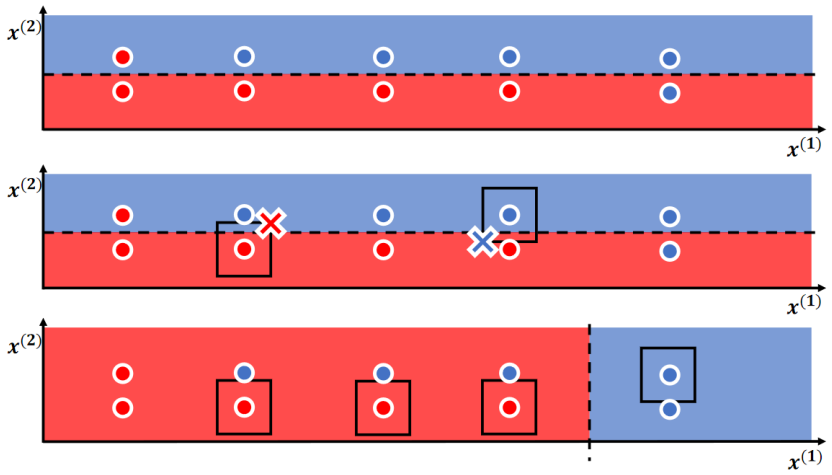


Figure 2.3: Motivating example from the work by Chen et al. [10]. A standard decision tree learning algorithm will choose the split in the top figure. However as we can see in the middle figure, an attacker can move samples within a box shape (L_∞ radius) and thereby cause every point to be misclassified. The bottom figure shows the robust split from Chen's algorithm, which is unaffected by the moving points.

2.3.1. Fitting Robust Decision Trees

We summarize the first work in the area of fitting adversarially robust decision trees and give extra attention to robust trees [10] and TREANT [7]. In Table 2.1 we compare each algorithm's time complexity and threat model capabilities. Robust decision tree works generally follow a white-box threat model with an attacker that aims to perturb samples at prediction time to cause misclassification.

Hardening decision trees Setting the foundations of adversarially robust decision trees, Kantchelian et al. [22] propose a hardening (adversarial training) approach for tree ensembles. They fit an ensemble similar to gradient boosting where each tree is trained both on the regular samples and on adversarial examples that evade the ensemble. They show that finding adversarial examples under the constraint of a distance function 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 accurately represent the threat model at test time. The hardening approach will only generate robust ensembles, so it can not generate single robust decision trees. Although we could extract one tree from the ensemble, this tree would likely not be very robust as in gradient boosting each tree predicts the residual of the earlier trees. Therefore it is the combination of trees that makes the ensemble robust and not the individual trees.

Robust decision trees against adversarial examples Instead of adding adversarial examples to the training set one can also directly fit a more robust model. Chen et al. [10] assume an attacker that is restricted by an L_∞ norm and give an algorithm to fit a tree that is more robust to these attacks (see Figure 2.3). To judge the quality of splits they use the information gain under attack. This value is the information gain for a situation in which the attacker moves points within an L_∞ radius to confuse the classifier maximally.

In general terms Chen's algorithm finds the best robust split, moves points near the split over it if that reduces the model's accuracy 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_∞ threat model.

To compute the information gain under attack Chen et al. iterate over all samples that can move over a split and find the configuration of samples that minimizes the information gain. Since this procedure needs to be repeated for every possible split fitting an entire tree can take a considerable amount of time. Therefore to train robust ensembles they speed up the algorithm by using a heuristic for the information gain under attack. In this heuristic they choose one of four options: do not move samples, move all samples left, move all samples right or swap the movable left and right samples. The heuristic can be computed in constant time

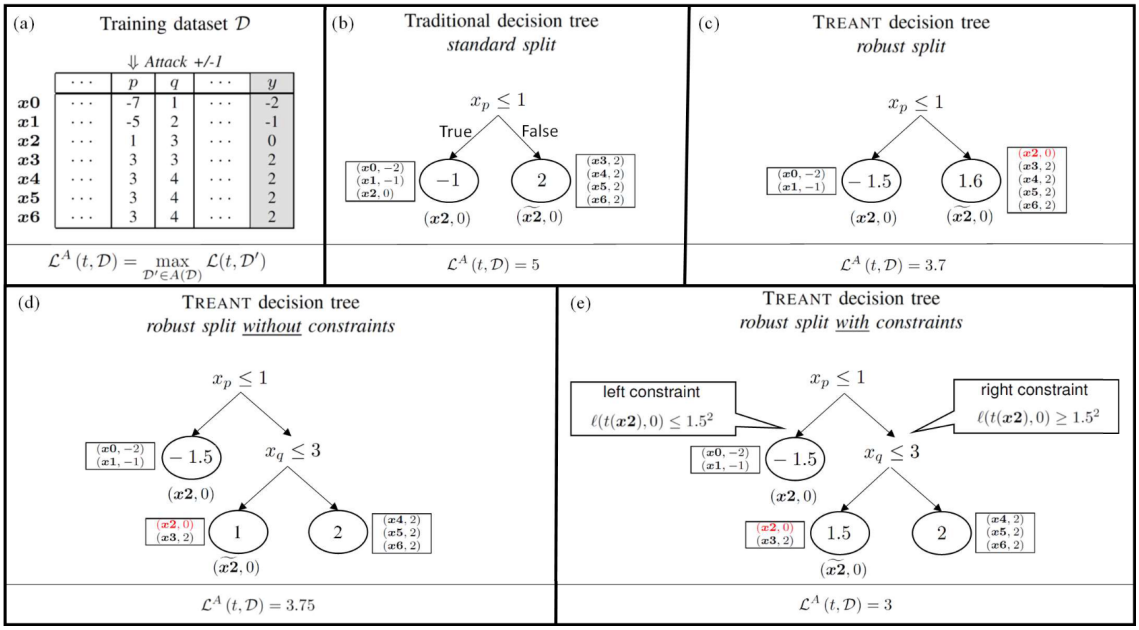


Figure 2.4: Summary of TREANT's learning procedure from [7]. To evaluate a split all samples are first optimally moved by an attacker (c). The algorithm ensures attack invariance which means that deeper trees cannot become less robust.

Table 2.1: Overview of algorithms for fitting adversarially robust decision trees. Where n is the number of samples, f number of features, d depth of the tree, r number of axis-aligned rules, X the solver time and i the number of hardening iterations. Recent works move away from L_x norm perturbations to support more realistic scenarios, all works assume a white-box scenario.

Name	Fit time per node	Threat model perturbations
TREANT [7]	$\mathcal{O}(n^2\, 2^d\, f\, 2^r)$	axis-aligned rules
Robust decision trees [10]	$\mathcal{O}(n^2\, 2^d\, f)$	L_∞
MILP hardening [22]	$\mathcal{O}(n \log n\, X)$	L_0 / L_1 / L_2 / L_∞
Approximate hardening [22]	$\mathcal{O}(n \log n\, i\, d 2^d)$	L_0

but it does not offer approximation bounds.

TREANT Previous works have assumed threat models in which the attacker was bounded by an L_x radius, the TREANT algorithm [7] however introduces a less restrictive concept. By allowing users 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 then directly optimizes a loss function under attacker influence (see Figure 2.4 for a visual demonstration) instead of using a splitting criterion like in regular decision trees or Chen's robust decision trees. So for each candidate split they compute the optimal attacks and score the split by a loss function given those attack occurred. Although this allows TREANT to train robust models against a wide variety of attackers, their algorithm deploys a solver to optimize their loss function and pre-computes in exponential time all the possible attacks which means the algorithm is slow in practice.

Additionally TREANT introduces attack invariance, a property that ensures trees can never become less robust when growing to a higher depth. This property improves the soundness of the algorithm.

2.3.2. Testing Robustness

Previous works have not agreed on one way to test robustness of decision tress. Kantchelian et al. [22] and Chen et al. [10] determine that to increase robustness is to increase the average distance (over an L norm) that samples have to move to be misclassified. In other words they attempt to increase the expected perturbation size necessary to misclassify samples. TREANT [7] computes metrics such as accuracy, f1 and area under the ROC curve (ROC AUC) after samples are perturbed by a specific threat model. We argue that these

Table 2.2: Metrics in the field of fair machine learning. Discrimination was originally used but research on predicting recidivism has pushed the field in the direction of equality of odds. All fairness constraints aim to enforce equality of a statistical property between groups.

Name	Formulation	Description
Non-discrimination	$\overline{y_p} = \overline{y_{\neg p}}$	Equal average prediction between groups [20, 21]
Consistency	$y = \frac{1}{k}\sum_{\text{nn}}^{k} y_{\text{nn}}$	Similar predictions for k near neighbors [39]
Equality of Opportunity	$FNR_p = FNR_{\neg p}$	Equal false negative rates [19]
Equality of Adversity	$FPR_p = FPR_{\neg p}$	Equal false positive rates (our proposal)
Equality of Odds	$X_p = X_{\neg p}, X \in \{FNR, FPR\}$	Equality of Opportunity and Adversity [38]

latter metrics better represent the kind of robustness that we aim to optimize. This is because these metrics represent how successful an adversary with specified threat model is against the model, in contrast with the expected perturbation distance metric that assumes that attackers are unbounded in their attacks.

Whereas the previous metrics can all be efficiently computed on decision trees by enumerating their leaves, measuring robustness of tree ensembles becomes increasingly costly. In their same work on hardening decision trees Kantchelian et al. [22] prove that finding adversarial examples in tree ensembles is NP-hard by a reduction from 3-SAT. Therefore they give a MILP formulation for this task and generate adversarial examples this way. Recent work attempts to speed up robustness verification by approximation [11].

Robustness verification Since determining robustness of tree ensembles using a MILP optimizer is extremely expensive Chen et al. [11] research whether they can verify the robustness of tree ensembles using an approximation. First they attempt to use a linear relaxation for the MILP formulation and find that the resulting lower bounds are loose. They then realize that the problem of finding adversarial examples in tree ensembles is equivalent to finding cliques in a weighted multipartite graph. Using an efficient algorithm they can then approximate the robustness and iterate this procedure to reach the optimum.

2.4. Discrimination in machine learning

With its increasing popularity, machine learning is more and more commonly used in sensitive applications. For these sensitive applications such as criminal recidivism prediction; non-discrimination is a vital property. We introduce the field of fair machine learning and discuss important related works.

Classifying without discrimination In 2009 Kamiran and Calders [20] kicked off the field of fair machine learning with their research on classification on biased datasets. They notice that when datasets contain a bias in racial attributes, these biases can easily propagate to a model that trained on the data. To remove this bias they massage the dataset and show that models learned from this new dataset show lower discrimination. Interestingly, they also find that just removing the sensitive attribute from the dataset is not as effective at removing discrimination than their massaging approach. This is likely caused by the sensitive attribute being correlated with different attributes and is something follow-up works should consider. They measure discrimination as the difference in average prediction between the protected and unprotected groups therefore they optimize for group fairness.

Learning fair representations Zemel et al. [39] attempt to learn a model that is not only fair for groups but also to individuals. By using clusters that maintain equal ratios of sensitive / not-sensitive samples as features both groups and nearest neighbors enjoy similar predictions, therefore low discrimination and inconsistency. Importantly, they also note that there is a trade off in accuracy and fairness. Using hyperparameters they authors control the trade off by optimizing either for low discrimination or for the maximum difference in discrimination and accuracy. Zemel et al. do not only optimize for group fairness using the discrimination metric but also introduce a metric for consistency. This metric enforces individual fairness by favoring models that predict similar values for near neighbors.

After seeing the imbalanced datasets in this work we realize, however, the importance of using dummy classifiers in fair machine learning. An example of a dummy classifier is a simple model that always predicts the most common class which is by definition discrimination free. Since many works use imbalanced datasets

we argue that they should compare to a dummy classifier because high accuracy and low discrimination models are easy to achieve here.

Equality of Odds In more recent literature works often aim to achieve equality of odds rather than the equal average predictions (non-discrimination). This is because equal average predictions can easily be achieved by randomly predicting values for one group to compensate for the average. This can cause large differences in the number of false positives or negatives. Instead of constraining prediction values, equality of odds enforces equal false positive rate (FPR) and equal false negative rate (FNR).

Hardt et al. [19] note that it is difficult to achieve both equality of FPR and FNR. Since they consider a model that predicts access to something positive they determine it is most important that both protected and unprotected group have a similar rate of false negatives. This ensures both groups have equal rate of mistakenly not allowing them the 'opportunity', the authors call this equality of opportunity.

Later in this work we will focus on models that predict a negative trait. Therefore we propose that in this case we prioritize equal false positive rates and refer to this as equality of adversity.

Fairness through adversarial learning Wadsworth et al. [38] provide another interesting method to remove the correlation of protected attributes with the prediction value, namely by adversarial learning. Their method is similar to a Generative Adversarial Network (GAN) in that it trains two neural networks that oppose eachother in turns. In their case study of recidivism prediction their first network predicts the probability of recidivism. Meanwhile, the second network attempts to predict the protected attribute from the first network's predictions. By punishing the first network for high success in the second network the authors manage to train a neural network with high accuracy and equality of odds.

2.5. Fair Decision Trees

There are a multitude of works that attempt to increase the fairness of decision trees. We discuss two methods that replace the splitting criterion to one that stimulates the growing of fair trees and one method based on a MILP formulation.

Fair Splits and Relabeling After kicking off the field of fair machine learning, Kamiran et al. [21] propose the first method for training fair decision trees. Their method changes the splitting criterion to one that combines the gain in information for the prediction label and the gain in protected attribute. Then after fitting the tree they relabel the leaves to remove discrimination while minimally reducing accuracy. Their approach proved to be successful in trading off accuracy for fairness in decision trees.

In their article, Kamiran et al. propose multiple ways of combining the Information Gain with respect to the Class (IGC) and Information Gain with respect to Sensitivity (IGS) for the splitting criterion. Specifically they propose:

- IGC – IGS: Favors splits that are homogeneous in class but heterogeneous in protected attribute
- IGC/IGS: Trades off accuracy with discrimination
- IGC + IGS: Favors splits that are homogeneous in both class and in protected attribute

The first two criteria intuitively promote fair splits as leaves will be more heterogeneous in terms of the number of protected / unprotected samples. In other words: leaves will less strongly favor one group over the other. Interestingly, they opt to use the last criterion. While this seems counter-intuitive, the criterion outperformed the other approaches in combination with their relabeling algorithm.

Fair Forests Raff et al. [34] present an ensemble method for training tree based fair classifiers called fair forests. They use the splitting criterion IGC – IGS that we saw in the work by Kamiran et al. [21] for training trees and combine them in a random forest. The fair forests achieve a discrimination and inconsistency score of 0.0 which means they are perfectly fair with respect to the protected attribute. However, a dummy classifier that always predicts the majority class will achieve the same accuracy and perfect fairness scores as the fair forests. Therefore we believe one should take caution when evaluating fairness on imbalanced datasets as the accuracy scores can be misleading.

Optimal Fair Trees Using a MILP formulation, Aghaei et al. [1] fit decision trees that optimally solve a trade-off between accuracy and fairness. Their formulation optimizes a loss function for accuracy and they add a regularization term weighted by a hyperparameter λ that punishes discriminating classifiers. Their resulting decision trees score well on accuracy while reducing fairness to almost 0%. Although the fitting decision trees using a MILP approach takes multiple hours for datasets up to thousands of samples, the authors rightfully note the generated decision trees can still predict efficiently.

2.6. Research Gap

Having evaluated the previous works in the fields of adversarial examples, decision tree learning and fairness we identified four research gaps:

- Many works used limited threat models. More permissible threat models such as those in TREANT [7] could not be intuitively formulated.
- Robust decision tree learning has been expensive. To the best of our knowledge all existing algorithms for training single robust decision trees scale by a complexity of at least $\mathcal{O}(n^2)$.
- Fairness and robustness have been studied separately but both could be required simultaneously. It is unknown to what extent this is possible.
- Many works opt to using ensembles as opposed to single decision trees but they have not always compared the two even though ensembles lose interpretability.

3

Robust Decision Trees Against Malicious Users

When we apply machine learning in adversarial settings we want the models to be interpretable, robust and accurate. In this chapter we introduce a more intuitive and permissible description for threat models based on TREANT [7]. Then we describe an efficient algorithm for training robust decision trees against these threat models similar to the algorithm by Chen et al [10]. After that, we introduce a hyperparameter ρ that we expect can facilitate a trade-off between accuracy and robustness. We compare our method to existing works and demonstrate its flexibility in learning against different threat models.

3.1. Specifying Threat Models

A threat model describes the assumed knowledge of an attacker and their attack capabilities. In this section we present a more intuitive and permissible threat model than the ones previous works use. When adversarial examples were introduced for perceptual models, the threat model was that all points could move within a L_x radius. Although this definition is convenient mathematically, the assumption that points can only move within this radius is unrealistic. In images for example the entire brightness or contrast can change with little visual change but large changes in the input vector. Additionally, for non-perceptual tasks features might be fixed in movement or only be movable in a specific direction.

Therefore to support a wide range of attack types we take inspiration from TREANT [7] and let the user define the space of possible perturbations for each feature individually. For this we use the following notation:

- “”or None: This feature cannot be perturbed.
- >: 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.
- A number ϵ: The feature can be perturbed in either direction by a distance of ϵ.
- A tuple (ϵ_l, ϵ_r): The feature can be perturbed ϵ_l to the left or ϵ_r to the 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 ϵ as (ϵ,ϵ). For conciseness we therefore only use the tuple notation in the algorithms in section 3.2. When we set the threat model to ϵ for each feature, it behaves identically to an L_∞ norm. In that case our method performs nearly identically to the Gini impurity version of the work by Chen et al. [10] where they do not use a heuristic.

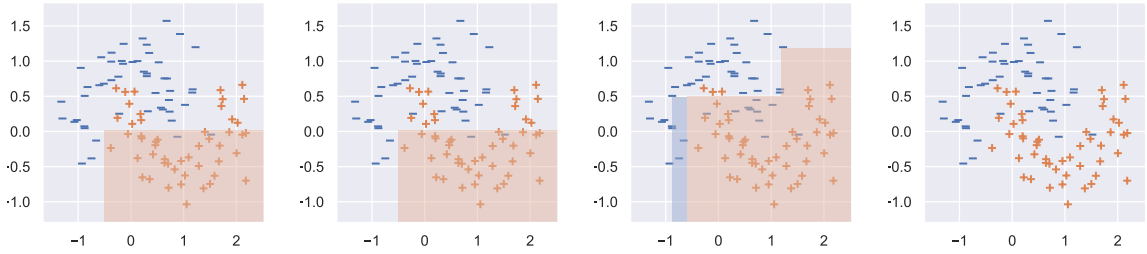
We introduce a separate notation for categorical features where the user defines a mapping from a category to a set of categories that it can be perturbed to. Say we have a feature that describes the language a person speaks, one might write the rule (English (UK) : {English (US), English (AU)}).

In this chapter we assume that we train models to distinguish between benign (class 0) and malicious (class 1) samples and that only malicious points can be moved. We do this since it would be impractical for a benign user to change its prediction to 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 misclassified as benign. We give examples of our threat models and highlight the importance of the threat model on robust decision tree learning in Figure 3.1.

Where TREANT proposes a budget to model attackers with different strengths we opt not to use this approach as it assumes arbitrary costs for perturbing features. We can do without this assumption by simply defining a new attacker for each strength with the notation we have presented.

To test the decision trees for robustness, we assume a white-box scenario in which the attacker knows everything about the model. To then quantify the model's robustness we enumerate all the tree leaves for every malicious test sample and see if we can reach a leaf of the opposite class. We then define robustness as the accuracy against an adversary that moved malicious points into benign regions to the best of its ability as described in the threat model.



(a) No attacker (b) (>, <) (c) (0.5, 0.5) (d) (<>, (0.7, 0.3))

Figure 3.1: Decision regions of trees trained with our method on a toy dataset that is attacked by different threat models (attack model is indicated below each image). The orange 'plus' samples move while the 'negative' samples are stationary. The type of threat model greatly influences robust decision tree learning. For instance, we see that decision trees that are robust against L_∞ perturbations (3.1c) are different from trees robust against more complex attackers (such as 3.1d).

3.2. Algorithm

Previous works show that although they are fast, decision trees trained greedily with the Gini impurity are vulnerable to adversarial attacks. To make this training method robust we revise the regular tree fitting algorithm by replacing the Gini impurity with the adversarial Gini impurity and propagate attacked samples to child nodes. Additionally, we augment the algorithm with a hyperparameter ρ to control the accuracy-robustness trade-off.

3.2.1. Scoring splits with the adversarial Gini impurity

We typically fit decision trees with a splitting criterion such as the Gini impurity that measures the probability of misclassification in a set. To determine the quality of a split we compute the Gini impurity on each side of the split and sum them weighted by number of samples on each side. We can define the Gini impurity for two classes as:

$$G(N_0, N_1) = 1 - \left(\frac{N_0}{N_0 + N_1}\right)^2 - \left(\frac{N_1}{N_0 + N_1}\right)^2 \tag{3.1}$$

Where N_0 and N_1 are the number of samples of class label 0 and 1 respectively. Then we combine this into a score function by taking the weighted average with respect to number of samples of the Gini impurity on each side of the split (other works refer to this function as the Gini gain):

$$S(L_0, L_1, R_0, R_1) = \frac{(L_0 + L_1) \cdot G(L_0, L_1) + (R_0 + R_1) \cdot G(R_0, R_1)}{L_0 + L_1 + R_0 + R_1}$$

$$S(L_0, L_1, R_0, R_1) = \frac{2\frac{L_0 L_1}{L_0 + L_1}}{L_0 + L_1 + R_0 + R_1} + \frac{2\frac{R_0 R_1}{R_0 + R_1}}{L_0 + L_1 + R_0 + R_1} \tag{3.2}$$

Where L_0 and L_1 are the number of samples on the left side of the split of label 0 (benign) and 1 (malicious) respectively. Similarly R_0 and R_1 represent the number of samples on the right side of the split. Normally one searches for a split that minimizes this score function, but this approach ignores the attacker capabilities. Instead, we will minimize a score function that is influenced by an attacker.

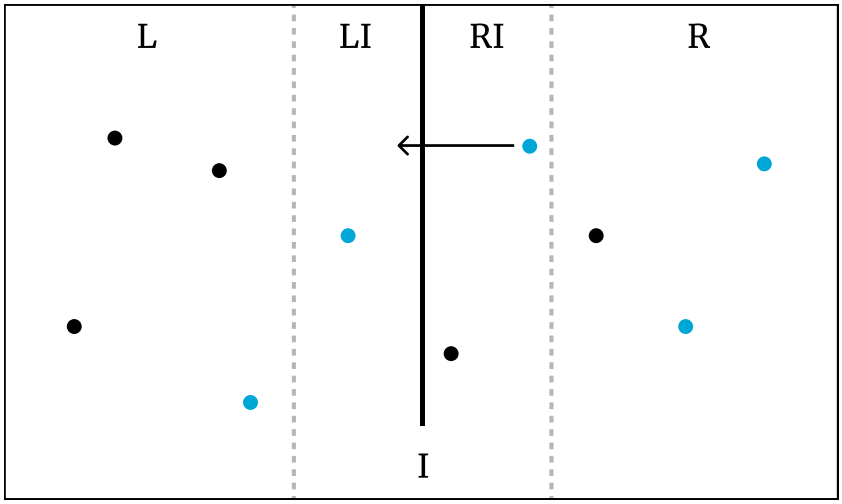


Figure 3.2: Visualization of the problem we are trying to solve when evaluating a split with the robust adversarial Gini impurity. The blue samples are malicious and can move within a user-specified range while the black samples are stationary. We want to move a number of samples from the intersection region (I) over the split such that the resulting Gini impurity is maximized. In this example we can move the single malicious sample from RI into LI to achieve the maximum Gini impurity on both sides. Where previous works [10] iterate over the samples in I, we find a constant time solution using the counts of samples in each region.

We define the adversarial score function as the worst case score function value that an attacker can cause. Where one normally minimizes the Gini impurity, we assume an attacker that aims to maximize $S(\cdot)$ by moving samples from I_1 to different sides of the split. Here, I_1 is the number of points with label 1 (malicious) that are close enough to the split that the adversary can move them to either side. We have visualized this maximization problem in Figure 3.2. Mathematically we are looking for the integer $x \in [0, I_1]$ so that x points of I_1 move to the left side of the split and $I_1 - x$ to the right. The score function under attacker influence is:

$$\begin{aligned} S_{robust}(L_0, L_1, R_0, R_1, I_1) &= \max_{x \in [0, I_1]} \quad S(L_0, L_1 + x, R_0, R_1 + I_1 - x) \\ &= \max_{x \in [0, I_1]} \quad \frac{2\frac{L_0(L_1+x)}{L_0+L_1+x}}{L_0 + L_1 + R_0 + R_1 + I_1} + \frac{2\frac{R_0(R_1+I_1-x)}{R_0+R_1+I_1-x}}{L_0 + L_1 + R_0 + R_1 + I_1} \end{aligned} \tag{3.3}$$

We can then write the x' that maximizes the score function under attacker influence as:

$$\begin{aligned} x' &= \arg\max_{x \in [0, I_1]} \quad \frac{2\frac{L_0(L_1+x)}{L_0+L_1+x}}{L_0 + L_1 + R_0 + R_1 + I_1} + \frac{2\frac{R_0(R_1+I_1-x)}{R_0+R_1+I_1-x}}{L_0 + L_1 + R_0 + R_1 + I_1} \\ &= \arg\max_{x \in [0, I_1]} \quad \frac{L_0(L_1 + x)}{L_0 + L_1 + x} + \frac{R_0(R_1 + I_1 - x)}{R_0 + R_1 + I_1 - x} \end{aligned} \tag{3.4}$$

The work by Chen et al. [10] has previously optimized a similar function (but in a setting where both classes move) by iterating through the I samples. However, since the function is concave with respect to x, we can do this faster by maximizing the function analytically and rounding the solution to one of the two nearest integers. Let us first simplify the equation for the maximum:

$$\frac{\partial}{\partial x}\left(\frac{L_0(L_1 + x)}{L_0 + L_1 + x} + \frac{R_0(R_1 + I_1 - x)}{R_0 + R_1 + I_1 - x}\right) = 0$$

$$\frac{L_0}{L_0 + L_1 + x} - \frac{L_0(L_1 + x)}{(L_0 + L_1 + x)^2} - \frac{R_0}{R_0 + R_1 + I_1 - x} + \frac{R_0(R_1 + I_1 - x)}{(R_0 + R_1 + I_1 - x)^2} = 0$$

$$\frac{L_0^2}{(L_0+L_1+x)^2} - \frac{R_0^2}{(R_0+R_1+I_1-x)^2} = 0 \tag{3.5}$$

When we solve this equation we find that the real number solution x' is:

$$x' = \frac{L_0R_1 + L_0I_1 - L_1R_0}{L_0+R_0} \tag{3.6}$$

We briefly prove that this solution is the real number maximum of Equation 3.4 by substituting it back into the left hand side of Equation 3.5:

$$\frac{L_0^2}{\left(L_0+L_1+\frac{L_0R_1+L_0I_1-L_1R_0}{L_0+R_0}\right)^2} - \frac{R_0^2}{\left(R_0+R_1+I_1-\frac{L_0R_1+L_0I_1-L_1R_0}{L_0+R_0}\right)^2} = 0$$

$$\frac{L_0^2}{\left(\frac{L_0^2+L_0R_0+L_0L_1+L_0R_1+L_0I_1}{L_0+R_0}\right)^2} - \frac{R_0^2}{\left(\frac{R_0^2+R_0L_0+R_1R_0+I_1R_0+L_1R_0}{L_0+R_0}\right)^2} = 0$$

$$\frac{1}{\left(\frac{L_0+R_0+L_1+R_1+I_1}{L_0+R_0}\right)^2} - \frac{1}{\left(\frac{R_0+L_0+R_1+I_1+L_1}{L_0+R_0}\right)^2} = 0$$

$$\left(\frac{L_0+R_0+L_1+R_1+I_1}{L_0+R_0}\right)^2 - \left(\frac{L_0+R_0+L_1+R_1+I_1}{L_0+R_0}\right)^2 = 0$$

$$0 = 0$$

Therefore x is the real number maximum of the function from equation 3.4. To prove that it is the only maximum we also prove that the function is concave by showing that the second order derivative is always negative. We continue by deriving the first order derivative we found in Equation 3.5:

$$\frac{\partial}{\partial x}\left(\frac{L_0^2}{(L_0+L_1+x)^2} - \frac{R_0^2}{(R_0+R_1+I_1-x)^2}\right)$$

$$= -\frac{2L_0^2(L_0+L_1+x)}{(L_0+L_1+x)^4} - \frac{2R_0^2(R_0+R_1+I_1-x)}{(R_0+R_1+I_1-x)^4}$$

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

From inspection we can see that this function will never be positive: all of our variables are greater or equal to zero and x is never larger than I_1. Therefore each fraction can only be positive and the minuses will cause the entire function to be negative.

As the function is concave with respect to x, the solution we found must be the only maximum. The integer solution is then either $\lfloor x' \rfloor$ or $\lceil x' \rceil$, which we can both efficiently test. Computing equation 3.6 and all the previous operations can be performed in $\mathcal{O}(1)$ time. Therefore we have an efficient method (constant time) for computing the Gini gain under attacker influence S_{robust}.

3.2.2. Finding the optimal split

Similar to regular decision tree fitting algorithms we can search over all possible splits and simply compute the adversarial Gini impurity to find a robust split. In Algorithm 1 we iterate over each sample (in sorted order) to identify candidate splits. We then evaluate each candidate split with the adversarial score function from the previous subsection to find the split that is most accurate against an adversary. We simply repeat this procedure for each feature to find the best overall split. However, 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 of both cases is bounded by $\mathcal{O}(n \log n)$ per feature.

Numerical case Differently than with a regular tree fitting algorithm, we need to not only consider splits at each sample but also splits at the points where a movable sample becomes in or out of range of moving over the split. Take for example a sample at position 3.3 that can be perturbed in a radius of 0.2, we test 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_0 / L_1 / R_0 / R_1 / I_1 at those splits, e.g. a sample moves from the right side of the split to an area where it can be on both sides of the split. We can evaluate a split in $\mathcal{O}(1)$ time by computing then rounding x' from equation 3.6 and 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 $\mathcal{O}(n \log n)$.

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 decision trees this way [14], we have not found such an algorithm for adversarial decision trees. Instead, 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 using a linear time pass over the data which means that the complexity for this step is $\mathcal{O}(n2^c)$ where n is the number of samples and c is the number of categories. The algorithm for the categorical case is given in Appendix A as it is very similar to the numerical case.

Algorithm 1 Find Best Robust Split on Numerical variable

function BESTROBUSTSPLIT(X) ▷ Input a set of observations
  $S \leftarrow X \cup \{o - \epsilon_l | o \in X_1\} \cup \{o + \epsilon_r | o \in X_1\}$ ▷ Identify candidate splits
  for $s \in S$ **do**
    $R \leftarrow \{o | o \in X_0 \wedge o > s\} \cup \{o | o \in X_1 \wedge o > s + \epsilon_r\}$ ▷ Sort samples in sets
    $RI \leftarrow \{o | o \in X_1 \wedge s < o \leq s + \epsilon_r\}$
    $LI \leftarrow \{o | o \in X_1 \wedge s - \epsilon_l < o \leq s\}$
    $L \leftarrow \{o | o \in X_0 \wedge o \leq s\} \cup \{o | o \in X_1 \wedge o \leq s - \epsilon_l\}$

    $l_0 \leftarrow |L_0|, \quad r_0 \leftarrow |R_0|$ ▷ Determine set sizes after ρ
    $l_1 \leftarrow |L_1| + (1-\rho)|LI|, \quad r_1 \leftarrow |R_1| + (1-\rho)|RI|$
    $i_1 \leftarrow \rho|RI| + \rho|LI|$

    $x'_s \leftarrow (l_0 r_1 + l_0 i_1 - l_1 r_0)/(l_0 + r_0)$ ▷ See equation 3.6
    $x'_s \leftarrow \min(\lfloor x'_s \rfloor, \lceil x'_s \rceil)$
    $g_s \leftarrow S(l_0,\ l_1 + x'_s,\ r_0,\ r_1 + i_1 - x'_s)$ ▷ See equation 3.2
  $s' \leftarrow \arg\max_s\ g_s$
  return $(s', g_{s'}, x'_{s'})$ ▷ Return best split, its score and optimal attack

3.2.3. Propagating samples

When fitting regular decision trees, the algorithm finds the best split and applies this rule to each sample to propagate each sample to the left or right side. Then the algorithm recurses and applies the same split search and sample propagation to the subset of samples on the left and right side until it reaches a stopping criterion. To account for samples that the adversary moves we make a slight modification to the propagation step as we define in Algorithm 2. We differ from the regular tree building algorithm by not only keeping track of left (L) and right (R) samples, but also storing a third set 'intersection' (I) that contains samples that can move to both sides. In section 3.2.1 we have shown that x' is the optimal value for the adversarial scoring function which was computed during the best split finding operation. Given the optimal number of left movable samples 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.

3.2.4. Trading off accuracy and robustness

We define accuracy as the accuracy score on the unperturbed dataset and adversarial accuracy (robustness) as the accuracy score on the dataset that is optimally perturbed by an attacker. To trade off between these two

Algorithm 2 Fit robust tree on numerical data

```
1:  function FitRobustTree(X)
2:      if pre-defined stopping criterion (e.g. maximum depth) then
3:          return Leaf(|X_0|, |X_1|)                          ▷ Create a prediction leaf
4:      else
5:          for f ← 1...F do
6:              s_f, g_f, x_f ← BestRobustSplit(X^f)           ▷ See Algorithm 1
7:          f' ← arg max_f g_f                                 ▷ Find the best feature to split
8:          determine R, RI, LI, L for split f'                ▷ See line 4 to 7 of Algorithm 1

9:          reduce |LI| to ρ|LI| by moving (1 − ρ)|LI| randomly chosen samples back to L
10:         reduce |RI| to ρ|RI| by moving (1 − ρ)|RI| randomly chosen samples back to R
11:         if |LI| < x_f' then                                ▷ Move samples in LI and RI for optimal x_f'
12:             move |LI| − x_f' randomly chosen samples from RI to LI
13:         else
14:             move x_f' − |LI| randomly chosen samples from LI to RI

15:         node_l ← FitRobustTree(L ∪ LI)                     ▷ Propagate samples to build the tree
16:         node_r ← FitRobustTree(R ∪ RI)
17:         return Node(s_f', node_l, node_r)
```

metrics, 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 malicious samples are created by attackers that will attempt an evasion attack. For example, if we use adversarial learning techniques for malware classification, we can expect that only a fraction of malware creators will change their code to avoid detection.

We make two changes to the algorithms to account for the parameter ρ. First, we scale the variable i_1 in Algorithm 1 to $\rho\, i_1$, so only the fraction ρ of malicious points can be moved by attackers. Second, during sample propagation (Algorithm 2) we reduce the size of the set I to $\rho|I|$ by random sampling so that only a ratio ρ of the samples move. The samples that were removed from I return to the side of the split where they belonged before perturbation (L_1 or R_1).

3.3. Results

We present results of normal decision trees, TREANT [7], and our algorithm on eight datasets in which we compare performance in a regular setting, adversarial setting and on run time. All datasets can be retrieved from openML and their specific versions are listed in Table 3.1. For regular decision trees we use scikit-learn's [31] implementation as it is widely used for research in the field. To compare against robust decision tree algorithms we run TREANT which outperformed both earlier works (hardening [22] and robust trees [10]). As our algorithm is near identical to robust trees [10] when setting the threat model to an L_∞ norm we do not compare against their work explicitly. We first compare the quality of the trees produced by each algorithm, then compare the run times and end by comparing robustness among different threat models.

3.3.1. Accuracy and robustness

To determine the quality of the algorithms we split each dataset in a train, validation and test set, each consisting of 80%, 10% and 10% of the samples respectively. First we optimize every algorithm's hyperparameters for accuracy in an adversarial setting on the validation set and then we run the best performing model on the test set. Of all these results we only report the test set scores. We run scikit-learn and our algorithm for tree depths 1 to 5 and TREANT only for depths 3 and 5 due to the algorithm's long runtimes. For scikit-learn we also vary the minimum number of samples required to make a split with the values (2, 20) or leaf with the values (1, 10, 20), for our algorithm we vary the value of ρ between 0 and 1 in 5 steps. The exact parameter settings can be found in our code[1] or in Appendix B. We were unable to run the TREANT algorithm on the ijcnn and cod-rna dataset due to exceeding memory limits (over 50GB after a day and no termination).

We define robustness as the accuracy on the test set after its malicious samples (class 1) have been op-

[1] Code will be made available online

Table 3.1: Accuracy and F1 scores before and after adversarial influence on the malicious class. Each algorithm was optimized for adversarial accuracy on a validation set then tested on a separate test set. Scikit-learn performs well in a regular setting but not in an adversarial setting. TREANT performs better but is outperformed in both settings by our algorithm.

	Scikit-learn				TREANT				Ours			
	Regular		Adversarial		Regular		Adversarial		Regular		Adversarial	
Dataset (openML name)	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1
census (adult (2))	**.844**	**.620**	.802	.461	.801	.415	.801	.415	**.844**	.616	**.843**	**.615**
cod-rna (codrnaNorm (1))	**.805**	**.633**	.637	.000	-	-	-	-	.669	.018	**.667**	**.001**
credit (default_credit_card_p (1))	**.817**	.435	.759	.093	.799	.297	.799	.297	.814	**.468**	**.814**	**.466**
diabetes (diabetes (1))	.610	.516	.571	.441	.649	.229	**.649**	.229	**.662**	**.536**	**.649**	**.509**
ijcnn (ijcnn (1))	.904	.000	.904	.000	-	-	-	-	**.910**	**.128**	**.906**	**.057**
ionosphere (ionosphere (1))	**.833**	.700	.778	.556	.778	.556	.667	.143	**.833**	**.727**	**.806**	**.667**
spambase (spambase (1))	.777	.646	.777	.646	.861	.810	.861	.810	**.883**	**.844**	**.874**	**.830**
wine (wine_quality (1))	**.735**	**.589**	.631	.314	.694	.455	**.694**	.455	.697	.477	.689	**.457**

timally perturbed by an attacker. Here an optimal perturbation means that if a sample can be moved to a benign region that perturbation will always be found. We calculate the optimal attack for a sample by enumerating all 2^d leaves, checking whether the leaf is in reach of sample and comparing the leaf's prediction to the actual label. The results of the experiment can be see in Table 3.1.

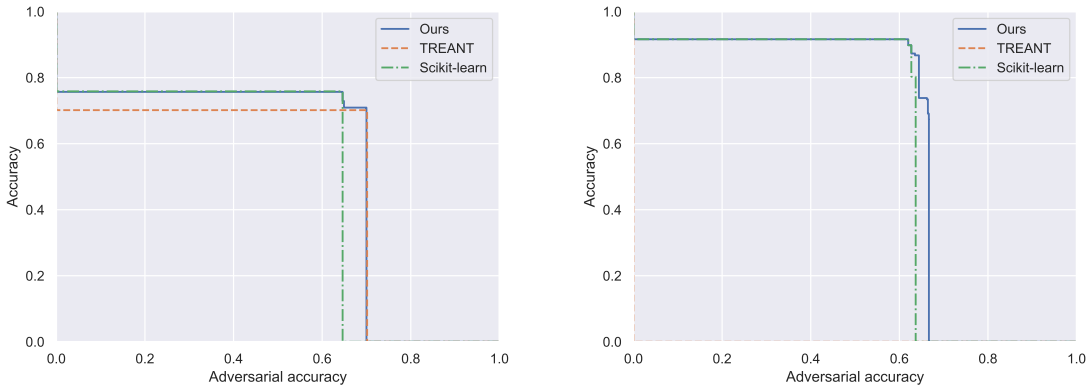


Figure 3.3: Outlines of the best achieved test scores for both accuracy and adversarial accuracy on the `wine` (left) and `cod-rna` (right) dataset among different parameter settings for each algorithm. Note how scikit-learn performs well on accuracy and TREANT well on robustness. Our algorithm performs well on both metrics. On `cod-rna` our algorithm also trades off different combinations of accuracy and adversarial accuracy.

From the resulting scores we see that while scikit-learn achieves good accuracy and TREANT achieves good robustness, our work matches scikit-learn's accuracy and significantly improves TREANT's robustness. Scikit-learn sometimes achieves an F1 score of 0.0 which happens when it learns to always predict the majority class. It is worth mentioning that TREANT assumes an attacker that moves samples from both classes and therefore is likely to fit a pessimistic model for this setting.

The scores from the table do not tell the entire story as there are also trees that have an accuracy and robustness in between these scores. For example, a tree can perform slightly worse on robustness but better on accuracy. To give a more complete overview of what combinations of accuracy and robustness the algorithms can achieve we also visualize the outlines of the test accuracy and robustness pairs on the wine dataset in Figure 3.3. We give visualizations for the other datasets in Appendix C.

We can interpret the lines as the capability of each algorithm to trade off accuracy and robustness. We again see that normal trees (scikit-learn) perform well at accuracy, that TREANT performs well at robustness and that our algorithm can perform well on both. Our algorithm performs well on both these metrics. Additionally, our algorithm can sometimes find multiple models that trade off accuracy and robustness to different degrees. For example on `cod-rna` there are different combinations of accuracy and robustness.

3.3.2. Run time

To compare the efficiency of the algorithms, we record the run time for fitting trees of depth five on each dataset. We run each algorithm five times on each dataset and visualize the results in logarithmic scale in Figure 3.4, we only run TREANT once due to long runtimes.

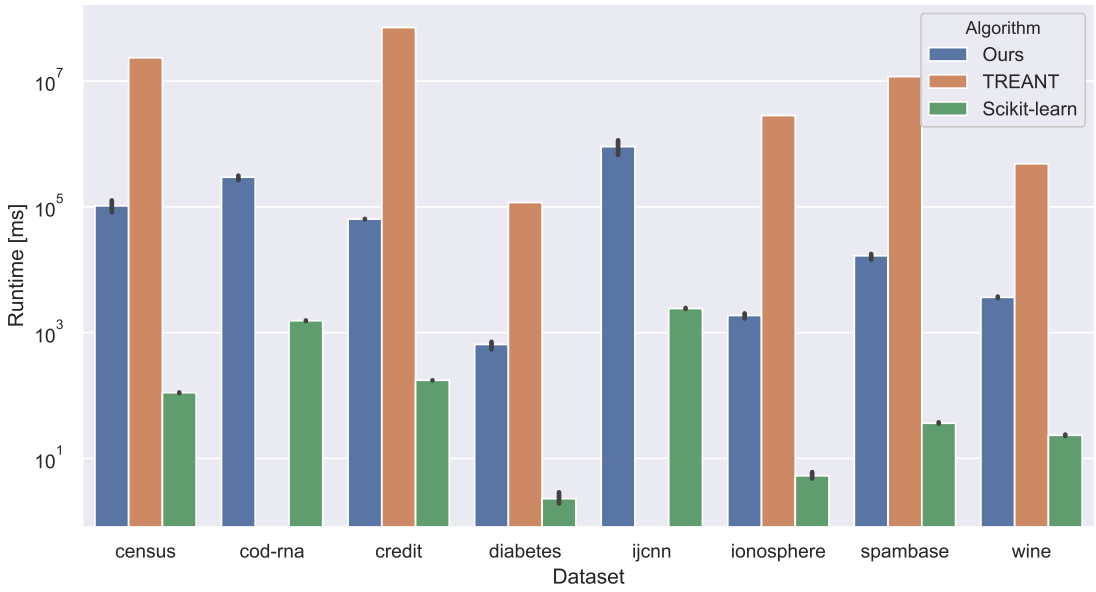


Figure 3.4: Run time comparison in milliseconds (on a logarithmic scale) for growing trees of depth 5 on different datasets. Our algorithm runs orders of magnitude faster than TREANT but still orders of magnitude slower than Scikit-learn. Algorithms ran on a single core on Google cloud machines with 6.5 GB of RAM per core. We could not get TREANT results for the cod-rna and ijcnn datasets as the process surpassed 30GB of RAM after running for multiple days.

Although the run time complexities for scikit-learn and our work are similar, the efficient C language implementation of scikit-learn runs very fast even for large sample sizes. It is worth noting that our algorithm will always be slower than scikit-learn as our method searches over more candidate splits. Comparing the robust algorithms TREANT and our work, we see that even though both are implemented in Python, our algorithm consistently runs two orders of magnitude faster.

3.3.3. Different threat models

To achieve good accuracy against adversaries it is important to fit models against that specific attacker, e.g. only training a tree to be robust to L_∞ perturbations does not yield high adversarial accuracy in all cases. We show this by defining five different threat models for the diabetes dataset and compare the performance of trees trained robustly on different threat models. We split the dataset in a 70% train and 30% test set and report the adversarial accuracy of each model tested against each threat model in Figure 3.5. The five threat models are: L_∞ norm bounded, the complex threat model used in the previous experiment, an attacker that modifies some (three) features freely but cannot modify the other features, all features can be increased in value, all features can be decreased in value. We find that the decision trees perform well against the threat model that they were trained on but they can perform significantly worse against others.

3.3.4. Effect of ρ

We introduced the hyperparameter ρ to control the accuracy-robustness trade off by moving a percentage of malicious points. Let us now investigate how the parameter ρ in combination with the tree depth actually affects model performance. We use the dataset splits from the previous experiment. Then we train decision trees until depth 5 to remain interpretable and vary the value of ρ. The resulting accuracy and robustness scores for `ionosphere` can be seen in Figure 3.6, figures for all datasets are given in Appendix F.

We see that increasing ρ usually decreases accuracy which can be explained by the fact that perturbing training samples causes the model to learn a different data distribution than the test set. We find that sometimes our parameter ρ is successful at trading off accuracy and robustness and other times increasing it low-

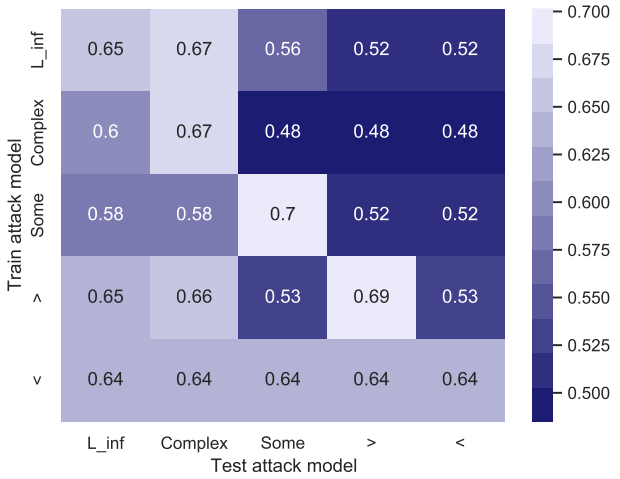


Figure 3.5: Comparison of adversarial accuracy scores of models trained and tested on different threat models for the diabetes dataset. The threat models are L_∞ norm bounded (L_inf), many different constraints (Complex), some features can be moved freely (Some), all features can be increased (>), all features can be decreased (<). We find that decision trees that are robust against one threat model often perform weakly against another.

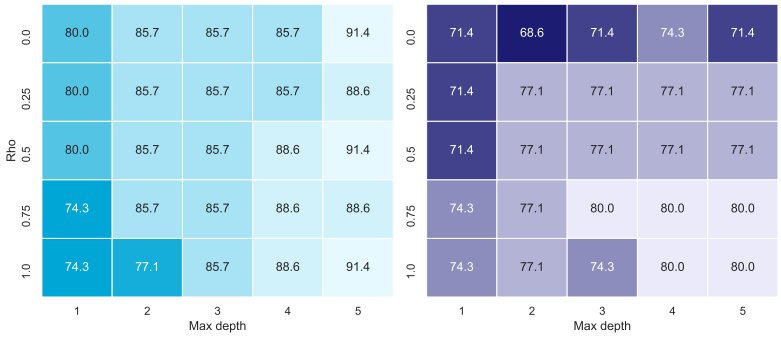


Figure 3.6: Accuracy (left) and robustness / adversarial accuracy (right) scores for the `ionosphere` dataset when varying the parameter ρ and the maximum depth of the tree. Users should vary both the tree depth and ρ since increasing depth can reduce robustness. Also, increasing ρ decreases accuracy. Since adversarial accuracy is limited by regular accuracy users should therefore experiment with different settings for ρ.

ers robustness. We hypothesize that since adversarial accuracy is always lower or equal to normal accuracy this explains why increasing ρ can sometimes decrease adversarial accuracy. Namely, the normal accuracy is affected by the moving points which in turn affects adversarial accuracy. We suggest this phenomenon is likely to occur for datasets that are weakly affected by an attacker as in that case the adversarial accuracy approaches normal accuracy.

Additionally we see that when increasing tree depth, robustness can decrease. Unlike TREANT our method does not assure 'attack invariance', this property enforces that deeper trees cannot decrease robustness. Therefore without attack invariance one should not only vary ρ but also tree depth to increase the quality of the results. We suggest users can use methods similar to the pruning procedures proposed by Breiman et al. [4] to limit the negative effects of large trees.

3.4. Discussion and conclusions

Our results have shown that our algorithm deploys an easy to use threat model, runs efficiently in time and can adapt well to different settings. It also performs on average 2.9% better in robustness, but it does not outperform TREANT in all cases. This might be due to TREANT's attack invariance that prevents trees from decreasing in robustness when they grow. In this work we limited ourselves to only moving malicious points but it would be interesting to train robust trees using an adaptation of our method that allows both benign

and malicious points to move.

We used accuracy and accuracy under adversarial influence to measure the performance of our models. Although the accuracy score is a common metric in the field it has issues capturing performance on imbalanced data. Future works could optimize for more specific metrics such as false positive rate or true positive rate that are important for use cases in security.

Another direction for future work might be to investigate the performance in a real-life use case as the current publicly available datasets are limited to improvised threat models.

In conclusion:

- We can train robust decision trees against a wide variety of attackers using an intuitive and permissible threat model formulation.
- By solving the Gini impurity under adversarial influence analytically we achieve a time complexity in terms of n samples of $\mathcal{O}(n \log n)$ instead of the $\mathcal{O}(n^2)$ complexity of other robust tree algorithms.
- Our method for training robust decision trees runs two orders of magnitude faster than the previous state-of-the-art work and consistently outperforms it in terms of robustness against attackers moving malicious samples.
- Where previous works were only accurate or robust, our algorithm can trade off these aspects using a user controlled hyperparameter ρ.

4

Robustness Against All Users

To the best of our knowledge, all previous works on robust decision tree learning [7, 10, 22] have assumed that all samples move. Therefore where we previously assumed only malicious users would perturb their inputs to a model to avoid detection, we will now assume both benign and malicious users want to evade the model. This will allow us to compare on similar settings against these works and to quantify the differences in performance against malicious and benign adversarial examples.

4.1. Adversarial Gini Impurity for Two Moving Classes

In this section we aim to find the analytical solutions to the maximization-based robust score function. We identify the solutions, prove they are the global maxima and show how we can efficiently turn this into an integer solution.

4.1.1. Summarizing Adversarial Gini Impurity for One Class

In the previous chapter we introduced the adversarial Gini impurity score function. This function scored the quality of a split and favored splits that separated benign and malicious samples well after attacker influence. The attacker influenced the function by moving malicious samples that were close to the split over it.

Let us first recall the adversarial Gini impurity score function and its simplified form:

$$S_{robust}(L_0, L_1, R_0, R_1, I_1) = \max_{x \in [0, I_1]} \quad \frac{2\frac{L_0(L_1+x)}{L_0+L_1+x}}{L_0+L_1+R_0+R_1+I_1} + \frac{2\frac{R_0(R_1+I_1-x)}{R_0+R_1+I_1-x}}{L_0+L_1+R_0+R_1+I_1}$$

$$= \arg\max_{x \in [0, I_1]} \quad \frac{L_0(L_1+x)}{L_0+L_1+x} + \frac{R_0(R_1+I_1-x)}{R_0+R_1+I_1-x}$$

Here L and R represented the number of samples on the left and right side of the split respectively and the underscores indicated the labels of the samples. I represented samples close enough to the split that they could be moved to either side by an attacker, hence I for 'intersection'. Since we only considered that malicious users (class 1) would move their samples we only used I_1. We also introduced the variable x that represents the number of samples from the intersection placed on the left side of the split. This leaves $I_1 - x$ malicious samples from I on the right.

Where previous works [10] set out to maximize a similar score function by iterating through the samples of I, we set out to solve this function analytically. By proving that the function was concave in the region of interest we identified the following solution:

$$x' = \frac{L_0 R_1 + L_0 I_1 - L_1 R_0}{L_0 + R_0} \tag{4.1}$$

This solution was then easily transformed to the integer solution by testing the rounded up and down integers $\lfloor x' \rfloor$ and $\lceil x' \rceil$. Note that if one would be interested in a setting where benign users attempt to evade the model one could use this function and simply flip all class labels.

4.1.2. Analytical Maximum for Two Moving Classes

Now let us take the simplified formulation for one moving class and add a variable y to it to take into account that both classes can move:

$$S_{robust} = \underset{x\in[0,I_1],y\in[0,I_0]}{\arg\max} \quad \frac{(L_0+y)(L_1+x)}{L_0+L_1+x+y} + \frac{(R_0+I_0-y)(R_1+I_1-x)}{R_0+R_1+I_0+I_1-x-y}$$

Notice how we also introduced the variable I_0 to count the number of benign samples close to the split. To find the maximum of this function we first find all critical points by setting both the derivative to x and the derivative to y equal to zero:

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

$$\begin{cases} \frac{L_0+y}{L_0+L_1+x+y} - \frac{(L_0+y)(L_1+x)}{(L_0+L_1+x+y)^2} - \frac{R_0+I_0-y}{R_0+R_1+I_0+I_1-x-y} + \frac{(R_0+I_0-y)(r_1+I_1-x)}{(R_0+R_1+I_0+I_1-x-y)^2} = 0 \\ \frac{L_1+x}{L_0+L_1+x+y} - \frac{(L_0+y)(L_1+x)}{(L_0+L_1+x+y)^2} - \frac{R_1+I_1-x}{R_0+R_1+I_0+I_1-x-y} + \frac{(R_0+I_0-y)(r_1+I_1-x)}{(R_0+R_1+I_0+I_1-x-y)^2} = 0 \end{cases}$$

This system of equations has three solutions that we will each examine, only one of which lies in our region of interest:

- $x = -L_1,\ y = -L_0$: 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, $y = \frac{L_1R_0 - L_0R_1 - L_0I_1 + L_1I_0 + (L_0+R_0+I_0)x}{L_1+R_1+I_1}$: 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:

$$x : \text{free}, \quad y = \frac{L_1R_0 - L_0R_1 - L_0I_1 + L_1I_0 + (L_0+R_0+I_0)x}{L_1+R_1+I_1} \tag{4.2}$$

When we plot the score function for some randomly chosen values for L_0, L_1, R_0, R_1, I_0 and I_1 we can get the impression that this line is indeed the maximum of the function (for $x \in [0, I_1]$, $y \in [0, I_0]$):

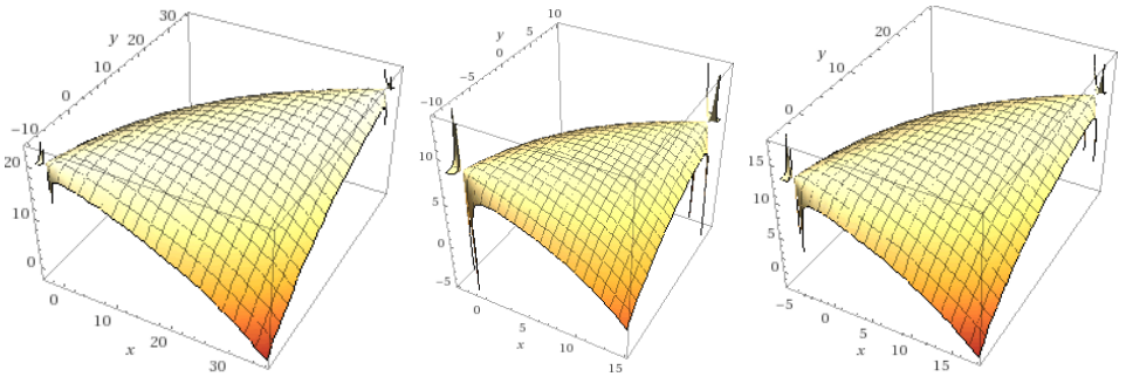


Figure 4.1: Function that we attempt to maximize for different settings of L_0, L_1, R_0, R_1, I_0 and I_1, plotted against x and y. The maxima appear to be lines through the diagonals of the plots.

To show that this line is the maximum of the function in the region of interest we computed the determinant of the hessian on the line. However this value turned out to be 0 so we need to analyze the function further to conclude whether or not it is a maximum. In our further analysis we will show that the function is concave in that region and therefore the critical points are maxima.

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:

$$H = \begin{bmatrix} \frac{\partial}{\partial x}\left(\frac{\partial}{\partial x} S_{robust}\right) & \frac{\partial}{\partial x}\left(\frac{\partial}{\partial y} S_{robust}\right) \\ \frac{\partial}{\partial y}\left(\frac{\partial}{\partial x} S_{robust}\right) & \frac{\partial}{\partial y}\left(\frac{\partial}{\partial y} S_{robust}\right) \end{bmatrix}$$

We continue by deriving the first order derivatives from the start of this section.

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

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

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

$$\frac{\partial}{\partial x}\left(\frac{\partial}{\partial y} S_{robust}\right) = \frac{\partial}{\partial y}\left(\frac{\partial}{\partial x} S_{robust}\right) =$$
$$\frac{1}{L_0+L_1+x+y} - \frac{L_0+y}{(L_0+L_1+x+y)^2} - \frac{L_1+x}{(L_0+L_1+x+y)^2} + 2\frac{(L_0+y)(L_1+x)}{(L_0+L_1+x+y)^3} +$$
$$\frac{1}{R_0+R_1+I_0+I_1-x-y} - \frac{R_0+I_0-y}{(R_0+R_1+I_0+I_1-x-y)^2} - \frac{R_1+I_1-x}{(R_0+R_1+I_0+I_1-x-y)^2} + 2\frac{(R_0+I_0-y)(R_1+I_1-x)}{(R_0+R_1+I_0+I_1-x-y)^3} =$$
$$2\frac{(L_0+y)(L_1+x)}{(L_0+L_1+x+y)^3} + 2\frac{(R_0+I_0-y)(R_1+I_1-x)}{(R_0+R_1+I_0+I_1-x-y)^3}$$

This leaves us with the hessian:

$$H = \begin{bmatrix} -2\frac{(L_0+y)^2}{(L_0+L_1+x+y)^3} - 2\frac{(R_0+I_0-y)^2}{(R_0+R_1+I_0+I_1-x-y)^3} & 2\frac{(L_0+y)(L_1+x)}{(L_0+L_1+x+y)^3} + 2\frac{(R_0+I_0-y)(R_1+I_1-x)}{(R_0+R_1+I_0+I_1-x-y)^3} \\ 2\frac{(L_0+y)(L_1+x)}{(L_0+L_1+x+y)^3} + 2\frac{(R_0+I_0-y)(R_1+I_1-x)}{(R_0+R_1+I_0+I_1-x-y)^3} & -2\frac{(L_1+x)^2}{(L_0+L_1+x+y)^3} - 2\frac{(R_1+I_1-x)^2}{(R_0+R_1+I_0+I_1-x-y)^3} \end{bmatrix} \tag{4.3}$$

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 function in the top-left corner of H:

$$-2\frac{(L_0+y)^2}{(L_0+L_1+x+y)^3} - 2\frac{(R_0+I_0-y)^2}{(R_0+R_1+I_0+I_1-x-y)^3} \tag{4.4}$$

Since the terms L_0, L_1, R_0, R_1 and $I_1 - x$, $I_0 - y$ 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:

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

$$\frac{\big((L_0+y)(R_1+I_1-x)-(L_1+x)(R_0+I_0-y)\big)^2}{(L_0+L_1+x+y)^3(R_0+R_1+I_0+I_1-x-y)^3} \tag{4.5}$$

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 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. We identified the only solution that lies in our region of interest, combining that with the concavity of the function means that we identified the maximum of the function in our region of interest. Therefore we can conclude that the maximum is the line:

$$x: \text{free}, \quad y = \frac{L_1R_0 - L_0R_1 - L_0I_1 + L_1I_0 + (L_0+R_0+I_0)x}{L_1+R_1+I_1}$$

4.1.3. Integer Maximum

In the previous subsection we have identified the real number maximum of S_{robust}, it was a line that we could easily compute. However when we want to use S_{robust} in our algorithm, we have to use an integer solution instead of the real number solutions for x and y. We could use one of two different approaches: iterate over all grid points surrounding the line to find the absolute maximum, or choose a grid point directly next to the line such that its value is close to the absolute maximum.

Chen et al. [10] have already described an iteration based approach for optimizing this function. In their approach they iterate through the samples in I and move samples to the side of the split that causes an increase in the value of the score function. This method is based on gradient ascent. Now that we have seen that the function is concave and the optimal values form a real number line, we know that their gradient ascent method indeed optimizes the function. However, note that they do not generally find the integer maximum but instead a grid point directly next to the line.

We will also aim for finding a grid point close to the line instead of finding the absolute integer maximum. We do this because it is fast and gives good approximation bounds. Instead of an iteration approach however, we want to round the analytical solution. As there are many integer solutions near the line we have to choose which one we want to find and we suggest the following:

1. Before moving samples over the split, there are starting values for x and y depending on how many samples of each class are close to the split.

2. Compute the point on the line that minimizes the distance to the starting values of x and y.

3. Round this point's location (x', y') to their nearest integer location $(\lfloor x' \rceil, \lfloor y' \rceil)$.

This way we intuitively choose the maximum which minimizes the number of samples to move. Additionally we get good approximation bounds as the distance between (x', y') and $(\lfloor x' \rceil, \lfloor y' \rceil)$ is at most:

$$\sqrt{\frac{1}{2}^2 + \frac{1}{2}^2} = \frac{1}{\sqrt{2}}$$

Computing Line Intersection To efficiently compute the closest point on the line to the starting (x, y) values we can use the fact that it lies on a second line perpendicular to the first line. Let us refer to the starting values as a, b. So we find the perpendicular line that passes through (a, b) 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 general 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 (a, b) is then [23]:

$$x' = \frac{-\left(-a - \frac{L_0+R_0+I_0}{L_1+R_1+I_1} b\right) - \frac{L_0+R_0+I_0}{L_1+R_1+I_1} \frac{L_1 R_0 - L_0 R_1 - L_0 I_1 + L_1 I_0}{L_1+R_1+I_1}}{\left(\frac{L_0+R_0+I_0}{L_1+R_1+I_1}\right)^2 + 1}, \quad y' = \frac{\frac{L_0+R_0+I_0}{L_1+R_1+I_1}\left(a + \frac{L_0+R_0+I_0}{L_1+R_1+I_1} b\right) + \frac{L_1 R_0 - L_0 R_1 - L_0 I_1 + L_1 I_0}{L_1+R_1+I_1}}{\left(\frac{L_0+R_0+I_0}{L_1+R_1+I_1}\right)^2 + 1}$$

$$x' = \frac{a + \frac{L_0+R_0+I_0}{L_1+R_1+I_1}\left(b - \frac{L_1 R_0 - L_0 R_1 - L_0 I_1 + L_1 I_0}{L_1+R_1+I_1}\right)}{\left(\frac{L_0+R_0+I_0}{L_1+R_1+I_1}\right)^2 + 1}, \quad y' = \frac{\frac{L_0+R_0+I_0}{L_1+R_1+I_1}\left(a + \frac{L_0+R_0+I_0}{L_1+R_1+I_1} b\right) + \frac{L_1 R_0 - L_0 R_1 - L_0 I_1 + L_1 I_0}{L_1+R_1+I_1}}{\left(\frac{L_0+R_0+I_0}{L_1+R_1+I_1}\right)^2 + 1} \tag{4.6}$$

Whenever we want to compute the S_{robust} we can now in constant time determine x', y' and round them to the nearest integers.

Edge cases Although we have chosen an intuitive method for selecting the integer solution, there are cases where this solution lies outside of the region of interest while there exists a solution inside it. Take for example the case in Figure 4.2.

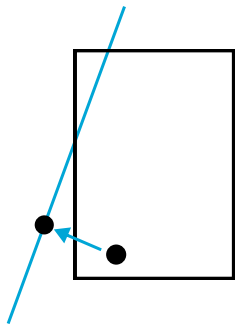


Figure 4.2: Example of a case in which the integer solution we find falls outside the region while the solution line does cross the region. The blue line represents the solution line, the point inside the region the starting position and the point after the arrow the integer solution. In this case we should choose the closest solution that is inside the region which is the intersection between the line and the region on the left side.

Here the integer solution lies outside the region while we could find a solution inside it. To solve this we can always check if the line intersects the region, then if it does and the integer solution is outside it we can test each intersection to find the closest one. These operations can all still be performed in constant time.

4.1.4. Relation to One Moving Class

Recall that in the previous chapter (Equation 3.6) we found an analytical solution for the case with one moving class, namely:

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

This same solution can be found from the more general formulation for two moving classes. For this we first rewrite the solution we found in this chapter by expressing x in terms of y:

$$x : \text{free}, \quad y = \frac{L_1 R_0 - L_0 R_1 - L_0 I_1 + L_1 I_0 + (L_0 + R_0 + I_0) x}{L_1 + R_1 + I_1}$$

$$x : \frac{L_0R_1 + L_0I_1 - L_1R_0 - L_1I_0 + (L_1 + R_1 + I_1)y}{L_0 + R_0 + I_0}, \quad y : \text{free}$$

Now since we are not moving benign samples we can simply include the samples from I_0 in their respective sides (L_0 or R_0). Then we can completely ignore y and I_0 as they are both 0 and we get the same solution that we found in the previous chapter:

$$x : \frac{L_0R_1 + L_0I_1 - L_1R_0 - L_1I_0}{L_0 + R_0 + I_0}$$

4.2. Results

We repeat the experiment from the previous chapter comparing accuracies and f1 scores but now in a setting where an adversary can move both classes. Although we already described the experiment setup, we will summarize it again.

To determine the quality of the algorithms we split each dataset in a train, validation and test set, each consisting of 80%, 10% and 10% of the samples respectively. We optimize every algorithm's hyperparameters for accuracy in an adversarial setting on the validation set and then we run the best performing model on the test set. For all models we vary the maximum depth, for scikit-learn we also vary the minimum samples required for leaves or nodes and for our work we vary the value of ρ. The exact methods can be found in our code[1]. Again, we were unable to run the TREANT algorithm on the ijcnn and cod-rna dataset due to exceeding time and memory limits.

In the adversarial setting an attacker has optimally moved all samples within the capabilities defined by the threat model. Here an optimal perturbation means that if a sample can be moved to a benign region that perturbation will always be found. We calculate the optimal attack for a sample by enumerating all 2^d leaves, checking whether the leaf is in reach of sample and comparing the leaf's prediction to the actual label. The results of the experiment can be see in Table 4.1.

Table 4.1: Accuracy and F1 scores before and after adversarial influence on both classes. Each algorithm was optimized for adversarial accuracy on a validation set then tested on a separate test set. Scikit-learn performs well in a regular setting but not in an adversarial setting. TREANT performs better but is outperformed in both settings by our algorithm.

	Scikit-learn				TREANT				Ours			
	Regular		Adversarial		Regular		Adversarial		Regular		Adversarial	
Dataset (openML name)	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1
census (adult (2))	**.843**	**.617**	.791	.448	.801	.415	.801	.415	.841	.605	**.840**	**.604**
cod-rna (codrnaNorm (1))	**.741**	**.672**	.037	**.062**	-	-	-	-	.667	.000	**.667**	.000
credit (default_credit_card_p (1))	**.817**	.435	.759	.093	.799	.297	.799	.297	.814	**.468**	**.814**	**.466**
diabetes (diabetes (1))	.610	**.516**	.494	**.400**	.649	.229	**.649**	.229	**.701**	.410	.597	.311
ijcnn (ijcnn (1))	**.904**	**.000**	**.904**	**.000**	-	-	-	-	**.904**	**.000**	**.904**	**.000**
ionosphere (ionosphere (1))	**.778**	**.556**	.583	.211	**.778**	**.556**	.444	.091	.750	.471	**.722**	**.444**
spambase (spambase (1))	.777	.646	.777	.646	**.861**	**.810**	**.861**	**.810**	.835	.774	.835	.774
wine (wine_quality (1))	.672	**.609**	.597	**.472**	.694	.455	.694	.455	**.697**	.460	**.695**	.459

Like in the previous chapter, we see that while scikit-learn achieves good accuracy and TREANT achieves good robustness, our work significantly improves TREANT's robustness. In contrast to the previous chapter where only one class moved, here we move two classes which is the setting TREANT has been designed for.

For clarity we have repeated Table 3.1 as Table 4.2 so that we can compare the scores between a setting in which one class moves versus a setting where both classes move. We find that the algorithms occasionally perform better in the case where only one class moves. This is what we expected as when a class' samples remain stationary they are intuitively easier to predict. We also see that TREANT's relative performance improves when going to the two moving classes setting which supports our claims from Section 3.3.3 that it is important to accurately define threat models for the test setting.

[1] Code will be made available online

Table 4.2: Accuracy and F1 scores before and after adversarial influence on a single class.

	Scikit-learn				TREANT				Ours			
	Regular		Adversarial		Regular		Adversarial		Regular		Adversarial	
Dataset (openML name)	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1
census (adult (2))	**.844**	**.620**	.802	.461	.801	.415	.801	.415	**.844**	.616	**.843**	**.615**
cod-rna (codrnaNorm (1))	**.805**	**.633**	.637	.000	-	-	-	-	.669	.018	**.667**	**.001**
credit (default_credit_card_p (1))	**.817**	.435	.759	.093	.799	.297	.799	.297	.814	**.468**	**.814**	**.466**
diabetes (diabetes (1))	.610	.516	.571	.441	.649	.229	**.649**	.229	**.662**	**.536**	**.649**	**.509**
ijcnn (ijcnn (1))	.904	.000	.904	.000	-	-	-	-	**.910**	**.128**	**.906**	**.057**
ionosphere (ionosphere (1))	**.833**	.700	.778	.556	.778	.556	.667	.143	**.833**	**.727**	**.806**	**.667**
spambase (spambase (1))	.777	.646	.777	.646	.861	.810	.861	.810	**.883**	**.844**	**.874**	**.830**
wine (wine_quality (1))	**.735**	**.589**	.631	.314	.694	.455	**.694**	.455	.697	.477	.689	**.457**

4.3. Discussion and Conclusions

In this chapter we have analyzed a robust score function that was previously computed using an iterative algorithm [10] and we found an analytical solution for it. We proved that the solutions lie on a line which means that there is not a single point in space that is optimal. The solution allows more efficient computation of the score function since it no longer requires iteration over n samples, instead it can be computed in constant time. In the previous chapter we limited ourselves to an attacker that could only move one class but with this more general solution we can also fit robustly against attackers moving both classes.

Since previous works [7, 10] assumed an attacker that moves both classes we could now test on that same setting. The results show that when comparing accuracy and f1 score, both in the unperturbed and adversarial setting, our algorithm in most cases outperforms TREANT.

We conclude that:

- Just like the setting in which one class moves, we can find a constant time solution to the score function for the setting in which two classes move.
- Using this formulation our algorithm in most cases outperforms TREANT on accuracy and f1 score in both the regular and adversarial setting.

5

Robust and Fair Trees

In literature robustness and fairness have been described as two desirable properties but they have been researched separately. We argue that for applications such as fraud detection we desire these properties simultaneous. In the chapter we combine the method for training robust decision trees from the previous chapter with a method to train fair trees. We want to test whether these methods combined can generate robust and fair trees. First we give our definitions for fairness, then describe our methods and then present the model's results.

5.1. Equality of Adversity

As we have seen in our literature review (Chapter 2) fairness in machine learning has been extensively studied but there is no unified definition of fairness. Earlier works often used the metric 'discrimination' or 'demographic parity', defined as the difference between average predictions for each group. However by enforcing these average predictions are equal we do not necessarily enforce fairness. For instance, a model might accurately predict one group but randomly select labels from the other as to maintain demographic parity.

Generally, we find that for fair classification we desire equality of odds. In equality of odds the following equation (from Hardt et al. [19]) should hold:

$$P\left(\hat{Y} = 1 | A = 0, Y = y\right) = P\left(\hat{Y} = 1 | A = 1, Y = y\right), \text{ for } y \in \{0, 1\}$$

Here Y is the class label, $\hat{Y}$ the predicted label and A the protected attribute. In other words: the true positive rate ($y = 1, \hat{Y} = 1$) and false positive rate ($y = 0, \hat{Y} = 1$) should be equal for both the protected and unprotected group. In their article Hardt et al. [19] argue they can focus on equality of opportunity. This means that they enforce equality of true positive rates between the groups but release the constraint for the false positive rates. They argue that since their models predict a positive or 'advantaged' outcome, and both groups should have the same odds for this opportunity.

Now in our case we predict a disadvantageous outcome, e.g. 'malicious' or 'fraud'. We argue that in this case equality of false positive rates ($y = 0, \hat{Y} = 1$) is the more important trait. Similarly, we saw that in the case of COMPAS the unfairness expressed itself mostly through false positives. Therefore in the rest of this chapter we will determine fairness mostly by the absolute difference in false positive rates between the two groups. We will refer to enforcing equal false positive rates as equality of adversity, defined as:

$$P\left(\hat{Y} = 1 | A = 0, Y = 0\right) = P\left(\hat{Y} = 1 | A = 1, Y = 0\right) \tag{5.1}$$

And we will measure how well a model adheres to this by measuring the absolute difference of false positive rates (FPR diff):

$$\text{FPR diff} = \left| P\left(\hat{Y} = 1 | A = 0, Y = 0\right) - P\left(\hat{Y} = 1 | A = 1, Y = 0\right) \right| \tag{5.2}$$

5.2. Robust and Fair Decision Trees

Let us now extend the notion of robust decision trees from the previous chapter to robust and fair decision trees. We will attempt to fit robust and fair decision trees by combining the adversarial Gini impurity with a

Figure 5.1: Accuracy, robustness and FPR difference for trees with different values of ρ and ϕ, maximum depth of 5 and run on `compas`.

splitting criterion that encourages fair splits.

Recall that Kamiran et al. [21] introduced three fair splitting criteria and a relabeling procedure which in combination fit fair trees (see chapter 2 for a discussion of this work). Since we attempt to fit trees without relabeling we opt to not use the IGC + IGS criterion but instead IGC – IGS as it encourages fair splits without relabeling. Raff et al. [34] also used this criterion for their work on fair forests.

Although this splitting function was created to improve the discrimination metric, we expect it will also improve the FPR diff metric. This is because when equal numbers of protected and unprotected samples are in a leaf they will all be predicted equally and thus have an equal FPR. Since the FPR over all samples is the average of the FPR for each sample this causes approximately equal FPR diffs among the groups. We can write the score function combining IGC – IGS and the score functions from the previous chapter as:

$$S_{fair}(L,R) = S(L_0,L_1,R_0,R_1) - S(L_{\neg p},L_p,R_{\neg p},R_p) \tag{5.3}$$

We will replace the left term by the robust score function from Equation 3.3. This function had an additional hyperparameter ρ associated with it to control the trade-off between robustness and accuracy. Recall that $0 \leq \rho \leq 1$ and that for $\rho = 0$ we get the regular non-robust score function. To control the trade-off with respect to fairness we aim to introduce a hyperparameter ϕ with the same properties. To achieve this we weigh the left and right terms by $(1-\phi)$ and ϕ respectively. the combined score function becomes:

$$S_{robust\&fair}(L,R) = (1-\phi)S_{robust}(L_0,L_1,R_0,R_1,I_1) - \phi S(L_{\neg p},L_p,R_{\neg p},R_p) \tag{5.4}$$

So now we have a score function in which we can stimulate more robust or fair splits by increasing the values of ρ and ϕ respectively. As we simply replace the score function from the previous algorithms we do not give the algorithms again.

5.3. Results

We will now test the performance of the combined formulation of fair and robust decision trees. First we perform a search over hyperparameter settings and compare the results with respect to accuracy, robustness and FPR difference. Then we further investigate the effect of accuracy on the levels of robustness and fairness that our models achieve. We test our methods on three datasets: `compas`, `bank_marketing` and `census`. All of these datasets are available at openML, more information on the datasets is given in Appendix D.

5.3.1. Model capabilities

We want to know if we can use the score function we proposed to train trees that are both robust and fair. To do this we will perform a hyperparameter search to attempt to find a model that has both these properties. We split stratify sample each dataset in an 80%-10%-10% train, validation and test set respectively and then perform the search over different values for ρ, ϕ and maximum depth. We plot the accuracy, robustness and FPR difference for the `compas` dataset in Figure 5.1. The figures for the other datasets can be found in Appendix E.

Table 5.1: Performance of our robust and fair decision trees when optimizing for different metrics. When optimizing for accuracy we perform almost identically to scikit-learn and when optimizing for fairness to a dummy classifier. By optimizing for accuracy + robustness under the constraint that FPR difference is lower than 2% (Ours - all) we improve both on the robustness and fairness of scikit-learn.

	compas			bank_marketing			census		
Model	Acc.	Rob.	FPR Diff.	Acc.	Rob.	FPR Diff.	Acc.	Rob.	FPR Diff.
Scikit-learn	64.6%	54.5%	16.7%	89.9%	85.3%	4.2%	84.4%	80.2%	5.8%
Ours - accuracy	65.2%	53.6%	9.6%	90.7%	85.9%	5.7%	84.1%	77.7%	5.2%
Ours - robustness	63.3%	63.3%	14.9%	89.5%	89.4%	4.0%	84.4%	84.3%	5.7%
Dummy	52.8%	52.8%	0.0%	88.3%	88.3%	0.0%	75.2%	75.2%	0.0%
Ours - fairness	52.8%	52.8%	0.0%	88.3%	88.3%	0.0%	75.2%	75.2%	0.0%
Ours - all	56.8%	56.8%	0.9%	88.8%	88.8%	0.4%	84.5%	84.5%	4.6%

We will optimize once for accuracy ($\rho = 0, \phi = 0$), robustness ($\phi = 0$) and fairness ($\rho = 0$) each. Then we will perform a more extensive search over all values of ρ, ϕ and maximum depth to find a decision tree that performs well on all three metrics. As it is often not practical to enforce equality of odds exactly, we will instead enforce the odds to be very similar for the last experiment. Specifically, we will search for decision trees that have an FPR difference of less than 5% and of these trees choose the one with maximum sum of accuracy and robustness. We list the results for each optimization procedure, Scikit-learn and a dummy classifier (always predicts benign) in Table 5.1.

The results show the versatility of our formulation for decision trees. When we optimize for accuracy we reach a score close to the accuracy of Scikit-learn. Similarly when we optimize for fairness we reach the same scores as a perfectly fair dummy classifier. Most importantly however, when optimizing for all metrics by searching over the full range of ρ and ϕ (Ours - all) the resulting tree has an FPR difference often far below 5% and is more robust than scikit-learn. This way our model is successful in finding trees that trade off all metrics.

An interesting result is that in the case of `compas` our fair and robust model only achieves an accuracy of 56.8% even after an extensive parameter search. The fact that the task of predicting recidivism is hard to do without maintaining fairness and robustness should make us think whether we want to use machine learning to perform this task. One could wonder whether it is even fair to make decisions on the basis of an inaccurate model.

5.3.2. Accuracy Correlation

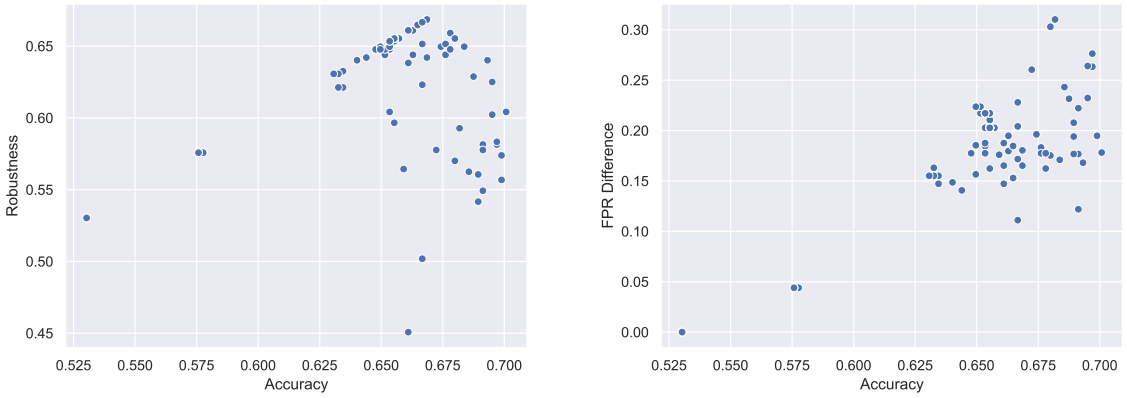
From Figure 5.1 we could already roughly see that accuracy and FPR difference are correlated, but we wish to investigate this property in more detail. Additionally we previously expected there to be a trade off between accuracy and robustness. Using the models trained during the parameter search for the dataset `compas` we visualize the accuracy-robustness and accuracy-FPR difference correlations in Figure 5.2. In these figures each point represents one model with specific parameter settings for ρ, ϕ and maximum depth.

When we inspect the relation of accuracy and robustness from Figure 5.2a we find a general upwards trend. Since attacks only move malicious points to decrease accuracy, robustness will always be lower or equal to the accuracy score. This is why we expect the clear diagonal line. Interestingly we can also see a trade off for accuracy > 0.66, even though we can achieve higher accuracy it lowers the robustness.

Where we want to maximize accuracy and robustness we want to minimize FPR difference. Therefore Figure 5.2b shows a more adverse result, increasing accuracy increases the FPR difference. Specifically we see fair results with varying accuracy between accuracy scores of 53% and 58% after which the fairness greatly deteriorates. Correlation plots for `bank_marketing` and `census` can be found in Appendix E.1.

5.4. Discussion and Conclusions

By combining an existing splitting criterion that promotes fair splits with our robust decision tree learning algorithm we were able to fit decision trees that are simultaneously fair and robust. We introduced a hyperparameter ϕ that trades off fairness and accuracy. Since our models make decisions that are adverse in nature we determined we would optimize fairness by enforcing equality of adversity. It would be interesting to see how our models perform on different fairness metrics such as discrimination, consistency and equality of opportunity.



(a) (b)

Figure 5.2: Correlation of robustness and FPR difference with accuracy on the `compas` dataset, each point represents the validation scores of a model from the hyperparameter search. We find from Figure 5.2a that robustness generally increases with increasing accuracy but it drops after a peak (67% accuracy here). After this peak we find a trade off in accuracy and robustness. In Figure 5.2b FPR difference appears to be positively correlated with accuracy, although there is a spread in points it seems that after 58% accuracy we get increasingly unfair models.

We conclude that:

- We can combine our robust learning algorithm with a fair splitting criterion to fit decision trees that are both robust and fair.
- Our combined algorithm trains trees with lower than 5% false positive rate difference between population groups, on average 6.9% less than Scikit-learn, while increasing robustness by on average 2.4%.
- FPR difference is correlated with accuracy, therefore when we only optimize for accuracy we will likely train unfair models.
- It is not always possible to train robust and fair models that are highly accurate. We should think about whether it is ethical to use machine learning to perform tasks that it cannot do accurately and fairly.

6

Tree Ensembles

As ensembles of decision trees have been very successful in machine learning lately, we can wonder if by using robust and fair decision trees we can improve ensembles as well. In this section we investigate the accuracy, fairness and robustness of random forests comprised of robust and fair decision trees. Specifically, we want to find out whether a random forest of robust and fair decision trees is also robust and fair. This is an assumption we believe is true as previous works have opted to using their robust decision trees in ensembles [7, 10]. Due to the increased complexity of finding adversarial examples in tree ensembles as compared to individual trees, we also go in depth on measuring ensemble robustness.

6.1. Robust and Fair Random Forest

To investigate the behavior of tree ensembles we will consider a random forest of the robust and fair decision trees that we have introduced in the previous chapter. One could also consider training gradient boosting ensembles but these methods require regression trees and our decision trees have been designed for classification. It would be interesting for future works to research a regression variant of robust and fair decision trees.

As discussed in the previous works chapter, random forests are an ensemble method published by Breiman [5] for decision trees. They use bootstrap aggregation and limit the number of candidate features per split to increase the diversity between the decision trees. We deploy these same methods and simply replace the regular decision trees by our parameterized robust and fair decision trees. For completeness we give pseudocode for the fitting procedure in Algorithm 3.

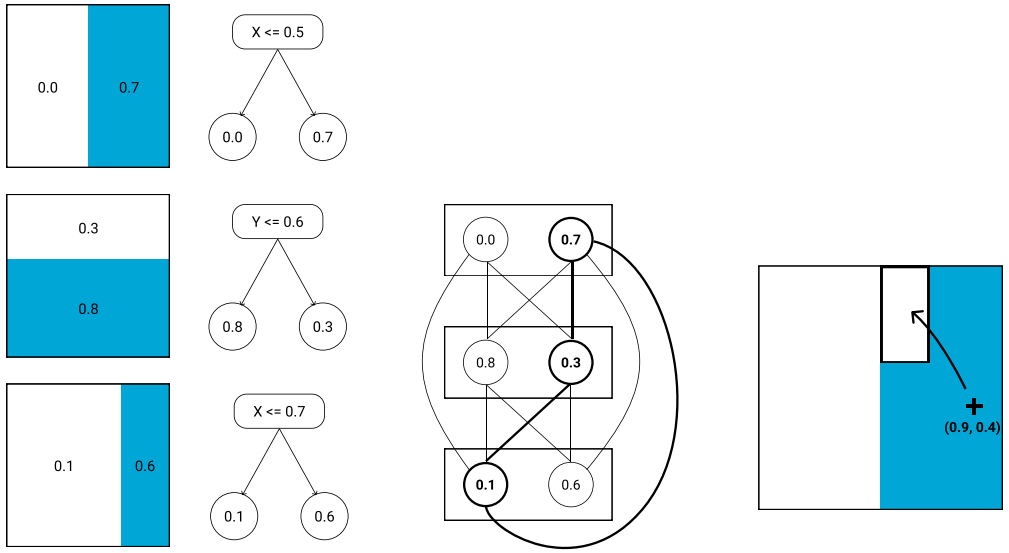
The entire tree fitting procedure from the previous chapter is the same except for the limitation of candidate features. What we mean by a limit to $\sqrt{f}$ of the features, is that for every split finding search we randomly sample $\sqrt{f}$ features and only search in these. This is to prevent many trees in the ensemble from using the exact same features to split on.

Algorithm 3 Fit Robust and Fair Random Forest

Input: X observations, Y labels, N, f number of trees and features, ρ, ϕ robustness and fairness trade offs

$RF \leftarrow \emptyset$
for $1...N$ **do**
    let X^* and Y^* be bootstrap samples of X and Y
    $t^* \leftarrow$ FitRobustFairTree($X^*, Y^*, \rho, \phi, \sqrt{f}$) ▷ Fit tree and limit splits to $\sqrt{f}$ candidate features
    $RF \leftarrow RF \cup \{t^*\}$

After fitting we can make predictions by averaging the predictions of the individual trees. As different works use different conventions it is important to note that we round after averaging the predictions instead of before (this is the same as scikit-learn). Therefore each tree predicts a value $0 \leq \hat{y} \leq 1$ and not a hard label 0 or 1. We weigh each tree equally. Combining all of this we get the prediction function for an input x:



(a) Ensemble of three decision trees and visualization of their decision regions.

(b) Graph encoding of ensemble, adversarial example highlighted in bold

Figure 6.1: Visualizations of a decision tree ensemble (blue is predicted malicious) and its encoding as a graph, nodes in the graph share an edge if the associated leaves intersect. Given an ensemble of three decision stumps we are trying to find an adversarial example for a malicious point at $(0.9, 0.4)$ that we assume here can move freely. We find a clique of average weight $\frac{0.7+0.3+0.1}{3} \approx 0.37$ which is less than 0.5 and therefore predicted as benign. So by moving the sample at $(0.9, 0.4)$ to e.g. $(0.6, 0.8)$ we find an adversarial example.

$$\hat{y} = \left\lfloor \frac{1}{N} \sum_{t \in RF} t(x) \right\rceil$$

6.2. Measuring Ensemble Robustness

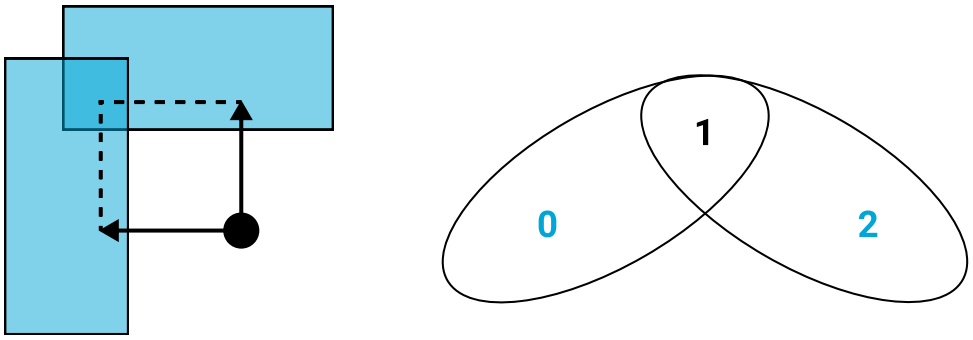
Where in single decision trees we can generate adversarial examples in polynomial time to compute the robustness, the problem becomes NP-hard for ensembles [22]. To still measure robustness in a reasonable time we use a method close to the one proposed by Chen et al. [11] in which they reduce the problem of finding adversarial examples to finding maximum cliques in a weighted multipartite graph.

This reduction can be made rather intuitively by encoding each decision tree as a partition of the multipartite graph and encoding each leaf as a node. Then we add a weight to each node equal to its corresponding leaf's prediction value (between 0 and 1) and we create an edge between a pair of nodes if these nodes intersect, i.e. if it is possible to have a sample that falls in both leaves. We give an example of a weighted multipartite graph encoding of a tree ensemble in Figure 6.1.

Now to find an adversarial example, we have to find a clique containing a node from each partition and compute the sum of weights of the nodes in that clique. If the average weight per node is lower than 0.5 the combination of leaves will predict benign and higher than 0.5 predicts malicious. So to find an adversarial example for a benign point we want to find a region that predicts malicious, an average weight of higher than 0.5.

Additionally, we need to check that the combination of leaves we found is in reach of the sample we are perturbing, according to the threat model. For ensembles containing only numerical variables we can build a graph with only the nodes that are in reach of the sample. This can speed up the search for cliques as it can significantly reduce the encoded graph size.

For ensembles that contain categorical variables, however, we need to take into account that if two leafs intersect and are each in reach of the sample their intersection is not necessarily in reach. Therefore when we build the graph we may only add an edge in between nodes after checking that the intersection of nodes is in the allowed perturbations by the threat model. We illustrate this extra condition in Figure 6.2.



(a) If numerical leaves are in reach, their intersection is also in reach

(b) If categorical leaves are in reach, their intersection might **not** be in reach

Figure 6.2: Visualization of the problem with categorical intersections. Reachable leaves and categories are blue. The intersection of reachable leaves with numerical conditions is always reachable for example in the case of Figure 6.2a. In the case of categorical conditions (sets of labels), the intersection might be out of reach such as in Figure 6.2b where a sample might be perturbable to the categories 0 or 2, but not to the category 1 in the intersection.

6.3. Results

We compare the quality of random forests with decision trees with respect to accuracy robustness and fairness on the dataset from the previous chapter. First we let both models fit on a grid of hyperparameter settings (ρ and ϕ) to examine the effect of the trade-off parameters in the case of a random forest. Where we restricted decision trees to a depth of 5 to preserve interpretability we let the ensembles of 10 trees train until depth 10 as we do not aim for interpretability here. We compare the accuracy scores of both models when we limit the FPR difference and optimize for accuracy + robustness. Lastly, we evaluate the performance compared to the level of interpretability.

6.3.1. Response to ρ and ϕ

To test the effect of the trade-off hyperparameters ρ and ϕ in the case of a random forest, we fit a forest and tree each on a large set of parameter settings. Specifically, we split the datasets `census`, `compas` and `bank_marketing` in a train, validation and test set with a 80%-10%-10% split. Then we train and validate each model on every combination of ρ and ϕ between 0 and 1 in steps of 0.2 keeping all other hyperparameters constant. We visualize the results separately for accuracy, robustness and fairness for the `compas` dataset in Figure 6.3.

Interestingly, we can see from these heatmaps not just the performance of the robust and fair trees but also the regular, robust or fair variants. When $\rho = 0, \phi = 0$ we get a regular decision tree similar to the Scikit-learn implementation [31], for $\rho = 0$ we get a fair variant and when $\phi = 0$ we have the robust trees from chapter 3.

6.3.2. Model capabilities

Now to more easily compare the performance of our decision trees and random forests we optimize once for each metric again and another time for all metrics simultaneously. We use the same optimization procedures from the previous chapter where when we optimize for all metrics, we maximize accuracy + robustness under the constraint that the FPR difference is less than or equal to 2%. The results are listed in Table 6.1. Again, for accuracy we set $\rho = 0, \phi = 0$, for robustness we vary ρ, for fairness we vary ϕ and for all we very both hyperparameters. All forests were trained with 10 trees and a maximum depth of 10.

We find that the two kinds of models perform similarly, where occasionally one outperforms the other. Since the differences occur on different metrics for different datasets we believe neither model generally improves over the other. Considering the greater runtime for forests than for a single tree and its impact on interpretability we suggest that decision trees are a more suitable model for the datasets considered. Additionally we compared the results of our trees and forests with the results from Scikit-learn's implementation and with a dummy classifier that is fair. We find that we can achieve very similar performance and that scikit-learn's forests also perform similarly to the trees.

Previous works have suggested that random forests are more robust to adversarial attacks [7, 10] and

(a) Robust and fair random forest (10 trees)

(b) Robust and fair decision tree

Figure 6.3: Validation scores for a random forest of ten robust and fair decision trees 6.3a vs a single robust and fair decision tree 6.3b (same as Figure 5.1) on the `compas` dataset. Although random forests appear to be more consistent with respect to accuracy their robustness is often lower and FPR difference higher than the single decision tree.

Table 6.1: Performance of our robust and fair random forests (10 trees) when optimizing for different metrics compared to robust and fair decision trees.

	`compas`			`bank_marketing`			`census`		
Model	Acc.	Rob.	FPR Diff.	Acc.	Rob.	FPR Diff.	Acc.	Rob.	FPR Diff.
Tree - accuracy	65.2%	53.6%	9.6%	90.7%	85.9%	5.7%	84.1%	77.7%	5.2%
Forest - accuracy	62.5%	50.0%	20.2%	91.2%	85.7%	7.5%	85.0%	80.3%	7.2%
Sklearn tree	64.6%	54.5%	16.7%	89.9%	85.3%	4.2%	84.4%	80.2%	5.8%
Sklearn forest	64.0%	48.9%	15.2%	90.5%	86.0%	6.4%	85.2%	81.0%	5.8%
Tree - robustness	63.3%	63.3%	14.9%	89.5%	89.4%	4.0%	84.4%	84.3%	5.7%
Forest - robustness	62.3%	60.2%	16.7%	89.2%	88.4%	11.8%	84.6%	84.5%	5.6%
Tree - fairness	52.8%	52.8%	0.0%	88.3%	88.3%	0.0%	75.2%	75.2%	0.0%
Forest - fairness	55.1%	55.1%	1.6%	88.5%	87.5%	0.3%	75.2%	75.2%	0.0%
Dummy	52.8%	52.8%	0.0%	88.3%	88.3%	0.0%	75.2%	75.2%	0.0%
Tree - all	56.8%	56.8%	0.9%	88.8%	88.8%	0.4%	84.5%	84.5%	4.6%
Forest - all	55.1%	55.1%	1.6%	88.4%	88.3%	0.0%	84.6%	84.4%	4.6%

more fair [34] than single decision trees. Our results do not confirm these suggestions. Instead our results align better with the work of Zhang et al. [40] who find that ensembles of support vector machines (SVMs) do not strictly outperform single SVMs.

6.4. Discussion and Conclusions

In this section we have evaluated the performance of random forests of our decision trees in part using a graph algorithm to determine robustness. We concluded that our random forests consisting out of 10 trees did not outperform the single decision trees. However, increasing the number of trees might affect this since it might increase consistency (reduce variance). Also, random forests can make use of more features than single decision trees so we expect their performance to be relatively better on datasets with many features. For future works it would be interesting to fit forests containing a mix of regular, robust and fair random trees to increase diversity and research whether this improves performance.

We conclude that:

- We can use our robust and fair decision trees to fit random forests that perform similarly to single trees.
- Although finding adversarial examples for random forests takes more run time, the forests are not generally more robust than decision trees.

7

Discussion

In this thesis we addressed the problem of training decision trees that are robust against user-specified adversarial attacks and fair to people of different population groups. The results show that by moving malicious samples with a user-specified threat model in the training procedure we can improve accuracy under attacker influence by on average 4.9% relative to regular decision trees. By speeding up the scoring of splits our algorithm runs two orders of magnitude faster than the state-of-the-art work on robust decision tree learning. Experiments on models that combine our robust trees with a splitting criterion that promotes fair splits show the possibility of training trees with around 1% false positive rate difference while improving robustness over regular decision trees.

These results show that it is possible to train decision trees that are more robust against adversarial examples and simultaneously fair. Additionally we hypothesized that the two hyperparameters that we introduced (ρ and ϕ) can trade off accuracy for robustness and fairness respectively. The results show that we can optimize our models for different combinations of accuracy, robustness and fairness and thereby support this hypothesis.

Our results suggest that there exist multiple decision trees that have different trade offs of accuracy and robustness that one cannot find by optimizing for only robustness or accuracy like previous works did. As all other works on robust and fair trees deployed their trees in an ensemble we expected to see an increase in performance when deploying our trees in a random forest. However, our results contradict this expectation as both models achieved similar scores.

7.1. Ethical Concerns

As we described algorithms for training models that can potentially be deployed in sensitive settings we need to discuss the ethical implications. In our research we referred to fairness as achieving equal false positive rates between population groups. Although this is a desirable property for predictive models, this description does not encompass everything that makes a model fair or unfair. There are a multitude of other metrics that one could argue should be equal between population groups, e.g. true positive rates, consistency and average prediction.

Additionally we want to warn that even when we manage to fit a fair model we need to consider the context it is deployed in. We need to think about whether we want to let a model make impactful decisions about human lives. In contexts like justice or policing we might require humans to make these decision on a case by cases basis. Our results show that it is not always possible to fit simultaneously accurate and fair decision trees, particularly when making impactful decisions we cannot afford high inaccuracy.

7.2. Limitations

We identify limitations regarding the generalizability of our results. In our entire research we assumed a white box threat model that always performs the worst-case attack. In practical cases an attack might have limited knowledge and will thus likely not perform worst case attacks. We optimized for accuracy and F1-score against these attackers but these metrics give limited information on true positive or false positive rates. Both these rates are important when deploying algorithms in a security setting.

Where we assumed attacker capabilities and attempted to reduce the evasion rates, another common approach is to assume an unbounded attacker and increase its perturbation distance. Since we only evaluated the former our method might not generate trees that perform well on both types of metrics.

In our implementation of decision trees we allow both numerical and categorical splits, where our categorical decisions split the categories in a left and right group. Other works only use a numerical encoding for categorical variables (scikit-learn [31], robust trees [10]) or send one category to the left and the other to the right (TREANT [7]). These design choices can affect the accuracy scores when limiting tree depth since e.g. splitting of categories one by one can require many decision nodes to make fine grained decisions. This way, a part of the performance improvements of our work might be caused only by the choice of splitting rules.

There are limited datasets in the field of adversarial machine learning. Although we reused the choices of threat models from other works were we could, the threat models and protected attributes were arbitrarily chosen for each dataset. Just like other works we worked with the class imbalanced versions of the datasets. The class imbalances can influence the usefulness of the accuracy and F1-score metrics when measuring model performance. For example, the minority class might be predicted poorly without affecting accuracy much.

For the comparison with ensembles of our decision trees and single trees we only tested random forests with 10 trees. As some works opt to use hundreds of decision trees in an ensemble, increasing the number of trees could improve the results. Some other works use gradient boosted decision trees ensembles instead of random forests which could also improve the results.

7.3. Recommendations

To establish the robustness of our method against black box attackers one could test our decision trees against boundary attacks, transfer attacks or other black box optimization strategies. It would be interesting to compare the improvement in evasion rates with the improvement in expected perturbation distance for unbounded attackers.

Our models have been shown to be flexible for different metrics by adjusting the hyperparameters ρ and ϕ. Further research could take into account different metrics such as TPR, FPR or fairness metrics like consistency and investigate the performance on those metrics.

By extending our decision trees to support regression tasks one could apply gradient boosting techniques. As, to the best of our knowledge, there are no previous works on trading off accuracy, robustness and fairness for tree ensembles it would be interesting to explore the capabilities.

8

Conclusions

This research aimed to find procedures to train decision trees that are robust against user-specified adversarial attacks and fair for different population groups. By moving samples over splits during training and incorporating a splitting criterion that promotes fairness we trained robust and fair decision trees. The results show that single shallow versions of these trees perform similarly to random forests of them.

We initially studied decision trees that are robust against user-specified adversarial examples. By analytically solving a score function that was previously iteratively solved we improved the time complexity of fitting robust trees to $\mathcal{O}(n \log n)$ over $\mathcal{O}(n^2)$ for n samples. We introduced a hyperparameter ρ that in agreement with our hypothesis facilitated models that trade off accuracy and robustness. This algorithm ran two orders of magnitude faster than the state of the art and improved predictive performance in an adversarial setting.

We found that this method was compatible with procedures from earlier works to train fair decision trees and combined the two methods to train robust and fair decision trees. To quantify the performance improvement of using these trees in an ensemble we compared it to a random forest. Contrary to our expectations this ensemble performed similarly to a single decision tree. While our assumption of a white box attacker and our optimization for accuracy / F1-score against an adversary limit the generalizability of our work, these results show new understanding of robust and fair decision trees.

8.1. Future Works

To better understand the effectiveness of the methods from this work, future work could test the performance against black box attack or optimize for different metrics. As our models proved to be flexible for performance on different metrics the trees could also potentially be used to create diversity in ensemble classifiers. Based on our conclusion that single trees performed similarly to ensembles, future studies should consider testing not only ensembles but also single decision trees.

Given the flexibility of the threat models that our algorithm can resist, future works can apply the methods to a variety of tasks. It would be interesting to test the models on different real life use cases such as detection of fraud, malware or malicious internet traffic.

8.2. Contributions

In our literature study we evaluated previous works on decision trees, adversarial robustness and fairness. We found a gap in work on efficiently fitting adversarially robust decision trees against user-specified threat models. Robustness and fairness were never considered simultaneously and tree ensembles were used over single trees with a lack of support. To address these gaps we contributed:

- An intuitive and permissible threat model for specifying attacker capabilities.
- An efficient algorithm for training robust decision trees against these threat models.
- An extension that fits simultaneously fair and robust decision trees.
- Hyperparameters ρ and ϕ that let models trade off accuracy, robustness and fairness.
- Comparisons between the performance of single decision trees and their ensembles with regards to robustness and fairness.

A

Robust split for categorical variable

We have previously described the robust splitting algorithm for numerical variables but for completeness we also give the categorical variant. The procedures are very similar only the way we define a split is different. Where a numerical split follows a $\leq$ for left and $>$ for right rule, a categorical split partitions the categories in a left and right set. For ease of notation we define $A(c)$ as the set of classes that condition c can be perturbed to. The function $A(c)$ can be derived from the notation for categorical variables as given in section 3.1. In that notation every category value maps to a set of categories that it can be perturbed to.

Algorithm 4 Find Best Robust Split on categorical variable

function BESTROBUSTSPLIT(X) ▷ Input a set of observations
  Let c_i be the number of elements of X that are equal to i
  $S \leftarrow \{c_0, c_1, ..., c_C\}$ where C is the number of categories
  for each partition (L, R) of S **do**
    for $c \in L$ **do**
      if $A(c) \cap R \neq \emptyset$ **then**
        $L \leftarrow L \setminus \{c\}$
        $LI \leftarrow LI \cup \{c\}$
    for $c \in R$ **do**
      if $A(c) \cap L \neq \emptyset$ **then**
        $R \leftarrow R \setminus \{c\}$
        $RI \leftarrow RI \cup \{c\}$
    $l_0 \leftarrow |L_0|, \quad r_0 \leftarrow |R_0|$
    $l_1 \leftarrow |L_1| + (1-\rho)|LI|, \quad r_1 \leftarrow |R_1| + (1-\rho)|RI|$
    $i_1 \leftarrow \rho|RI| + \rho|LI|$

    $x'_s \leftarrow (l_0 r_1 + l_0 i_1 - l_1 r_0)/(l_0 + r_0)$ ▷ See equation 3.6
    $x'_s \leftarrow \min(\lfloor x'_s \rfloor, \lceil x'_s \rceil)$
    $g_s \leftarrow S(l_0,\ l_1 + x'_s,\ r_0,\ r_1 + i_1 - x'_s)$ ▷ See equation 3.2
  $s' \leftarrow \text{argmax}_s\ g_s$
  return $(s', g_{s'}, x'_{s'})$ ▷ Return best split, its score and optimal attack

B

Robust Trees Experiment set-up

We summarize the eight datasets that we used in Section 3.3 and the threat models that we defined for them in Table B.1. For more details we refer to the attached code. The number of instances mentioned for each dataset represents the ones left after discarding instances with missing values. All datasets can be found on openML[1]. The specific names on openML are: `adult (2)`, `wine_quality (1)`, `default_credit_card_p (1)`, `diabetes (1)`, `codrnaNorm (1)`, `ionosphere (1)`, `ijcnn (1)` and `spambase (1)`.

In Table B.2 we give the parameter settings from the experiment. We run each algorithm on every combination of the given parameters.

Table B.1: Properties of the eight datasets that we used in our experiments along with a description of the kind of attack model we used. More details can be found in the attached code.

Dataset	#samples	#features	threat model
`census`	45222	14	Adapted from TREANT
`wine`	6497	11	Adapted from TREANT
`credit`	30000	23	Adapted from TREANT
`diabetes`	768	8	Measurements can be changed slightly
`cod-rna`	488565	8	L_∞
`ionosphere`	351	34	L_∞
`ijcnn`	191681	22	L_∞
`spambase`	4601	57	Word counts can be increased

Table B.2: Overview of the parameter settings that we used in our experiments. Each algorithm was run on each combination of its parameters.

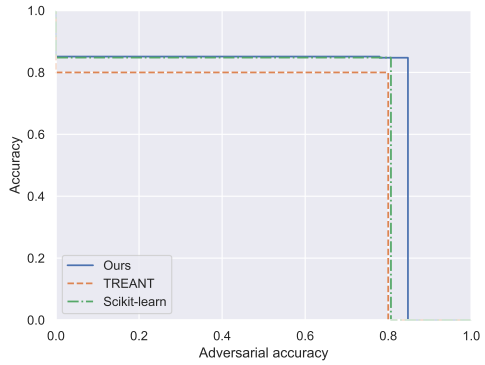
Algorithm	Parameter settings
Scikit-learn	{`max_depth`: 1...5, `min_samples_leaf`: [1,10,20], `min_samples_split`: [2,20]}
Ours	{`max_depth`: 1...5, ρ: [1,0.75,0.5,0.25,0]}
TREANT	{`max_depth`: [3,5]}

[1] `https://www.openml.org/`

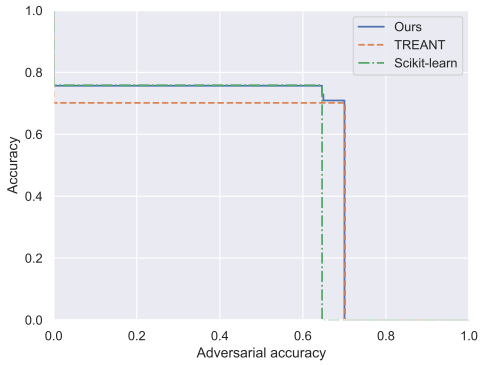
C

Accuracy robustness trade-off plots

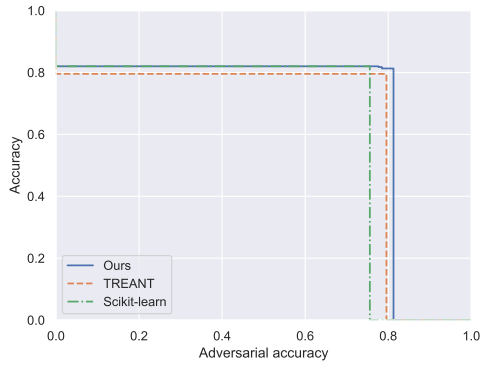
In Section 3.3 Figure 3.3 we visualized the trade-off between accuracy and robustness for the wine dataset. For completeness we give the trade-off plots for all datasets in Figure C.1. In all these models only malicious samples moved.



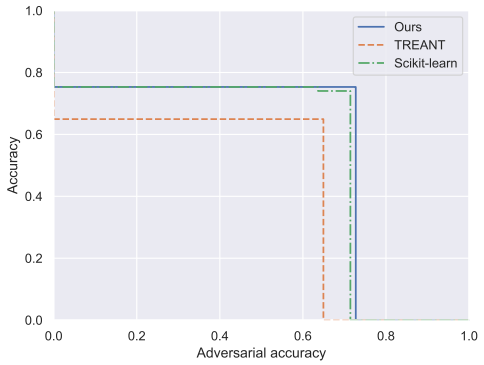
(a) census



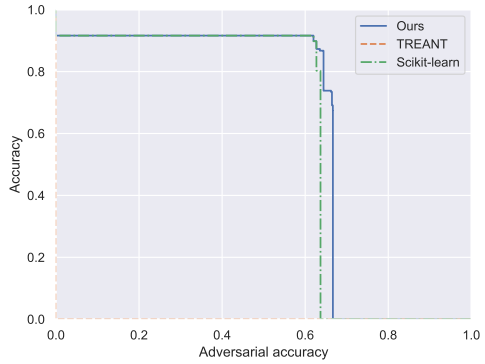
(b) wine



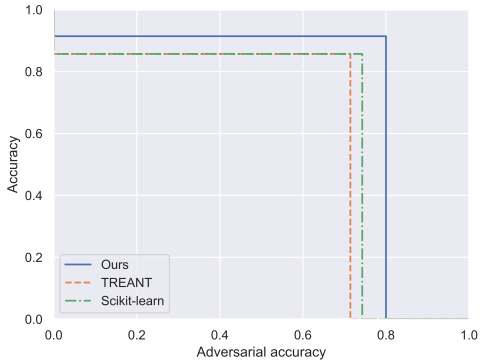
(c) credit



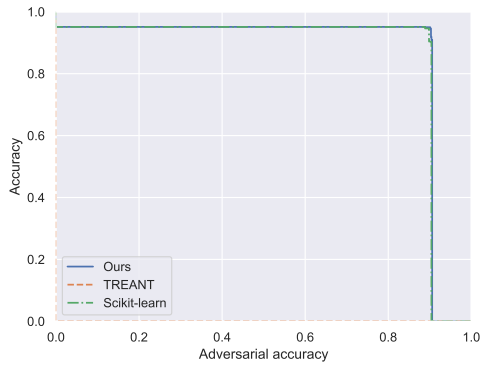
(d) diabetes



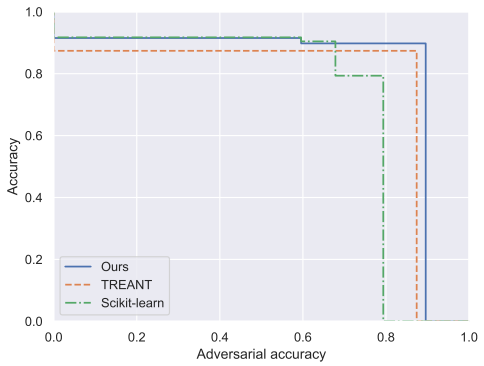
(e) cod-rna



(f) ionosphere



(g) ijcnn



(h) spambase

Figure C.1: Outlines of the best achieved test scores for both accuracy and adversarial accuracy on all datasets among different parameter settings for our robust tree algorithm, TREANT and scikit-learn.

D

Robust and Fair Trees Experiment set-up

We summarize the three datasets that we used in Chapter 5, the threat models that we defined for them and the protected attribute in Table D.1. For more details we refer to the attached code. The number of instances mentioned for each dataset represents the ones left after discarding instances with missing values. All datasets can be found on openML[1]. The dataset names on openML are: `compas-two-years (4)`, `bank_marketing (1)`, `adult (2)`.

Table D.1: Properties of the three datasets that we used in our robustness and fairness experiments along with a description of the kind of attack model we used and protected attribute. More details can be found in the attached code.

Dataset	#samples	#features	threat model	protected attribute
`compas`	5278	8	Age modified by 10 years	race (caucasian, african-american)
`bank_marketing`	45211	16	Change categories, duration <>	age (< 25 and > 60, > 25 and < 60)
`census`	45222	14	Adapted from TREANT	sex (male, female)

The `compas` dataset contains features on defendants' criminal history and aims to predict recidivism, the likelihood that that person will reoffend. COMPAS is a commercial algorithm that used this dataset to make predictions, but researchers from ProPublica[2] found unfairness between population groups. Particularly African American citizens who did not reoffend in the two following years were twice as likely to be wrongly predicted as high risk than Caucasian citizens.

The `bank_marketing` and `census` datasets contain multiple features on people's employment status, marital status and age. `bank_marketing` attempts to predict whether people will subscribe to a banking product after marketing. `census` attempts to predict whether people make over $50K$ per year.

[1] `https://www.openml.org/`

[2] `https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing`

E

Parameter Search ρ and ϕ

We give the heatmaps representing the parameter search over ρ and ϕ for the datasets `bank_marketing` and `census` that were not included in the main text. We also plot their accuracy versus robustness and accuracy versus FPR difference.

(a) Robust and fair random forest (10 trees)

(b) Robust and fair decision tree

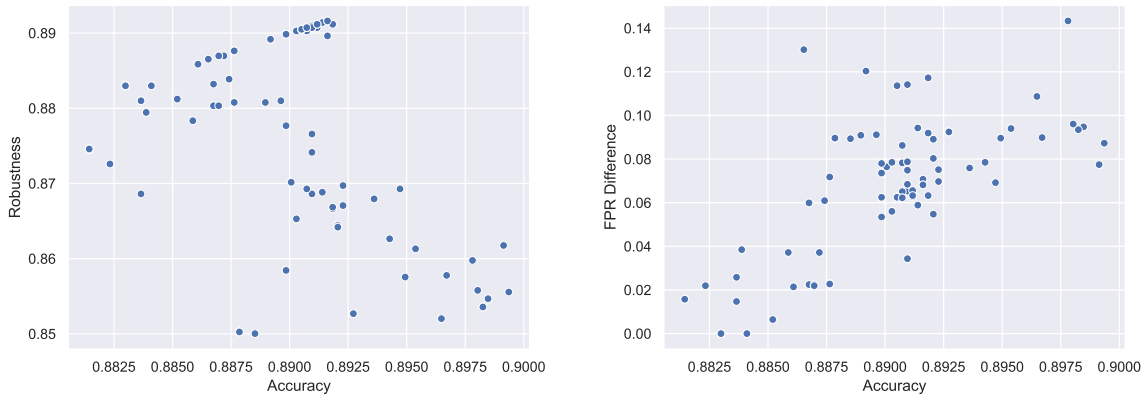
Figure E.1: Validation scores for a random forest of ten robust and fair decision trees vs a single robust and fair decision tree on the `bank_marketing` dataset.

(a) Robust and fair random forest (10 trees)

(b) Robust and fair decision tree

Figure E.2: Validation scores for a random forest of ten robust and fair decision trees vs a single robust and fair decision tree on the `census` dataset.

E.1. Correlations



(a) (b)

Figure E.3: Correlation of robustness and FPR difference with accuracy on the `bank_marketing` dataset, each point represents the validation scores of a model from the hyperparameter search.

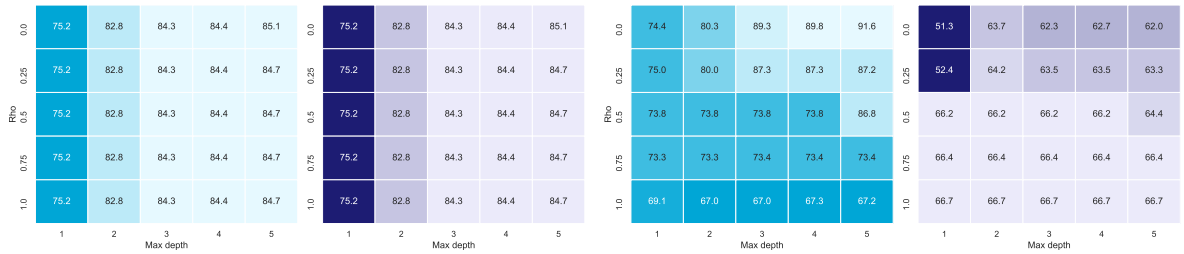
(a) (b)

Figure E.4: Correlation of robustness and FPR difference with accuracy on the `census` dataset, each point represents the validation scores of a model from the hyperparameter search.

F

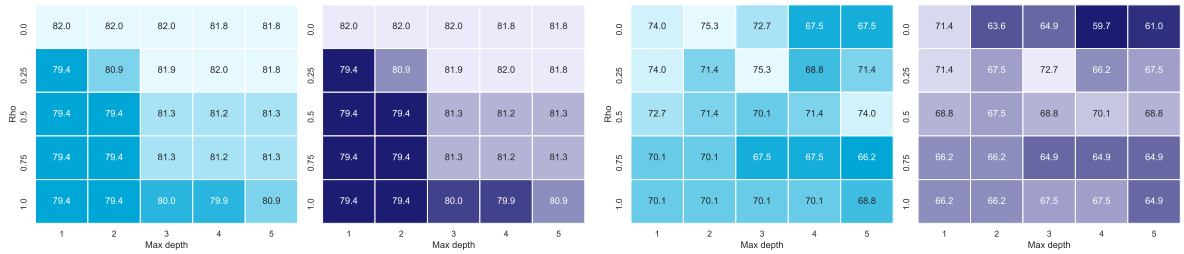
ρ and Tree Depth

In Chapter 3 we examined the combined effect of ρ and tree depth on accuracy and robustness. In the chapter we only visualized the `credit` dataset results, in this appendix we give the figures for all eight datasets.



(a) `census`

(b) `cod-rna`

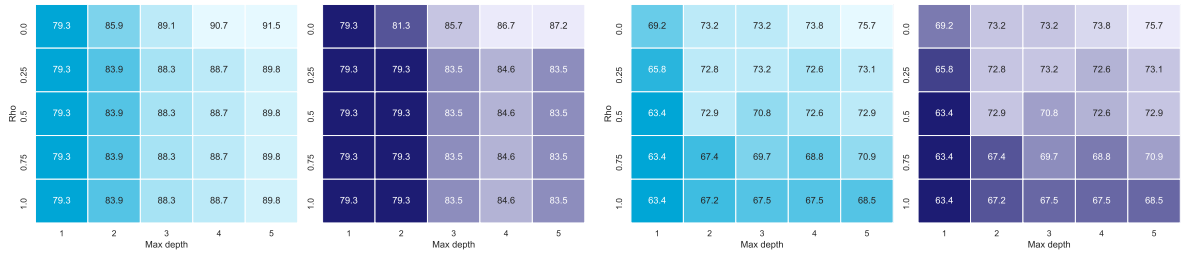


(c) `credit`

(d) `diabetes`

(e) `ijcnn`

(f) `ionosphere`



(g) `spambase`

(h) `wine`

Figure F.1: Effect of varying ρ and maximum depth hyperparameters on accuracy (blue) and adversarial accuracy (purple).

Bibliography

[1] Sina Aghaei, Mohammad Javad Azizi, and Phebe Vayanos. Learning optimal and fair decision trees for non-discriminative decision-making. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1418–1426, 2019.

[2] William A Belson. Matching and prediction on the principle of biological classification. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 8(2):65–75, 1959. Publisher: Wiley Online Library.

[3] Kristin P Bennett and Jennifer A Blue. Optimal decision trees. *Rensselaer Polytechnic Institute Math Report*, 214:24, 1996.

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

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

[6] 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.

[7] Stefano Calzavara, Claudio Lucchese, Gabriele Tolomei, Seyum Assefa Abebe, and Salvatore Orlando. Treant: Training Evasion-Aware Decision Trees. *arXiv:1907.01197 [cs, stat]*, July 2019.

[8] Nicholas Carlini and David Wagner. Magnet and" efficient defenses against adversarial attacks" are not robust to adversarial examples. *arXiv preprint arXiv:1711.08478*, 2017.

[9] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. IEEE, 2017.

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

[11] Hongge Chen, Huan Zhang, Si Si, Yang Li, Duane Boning, and Cho-Jui Hsieh. Robustness verification of tree-based models. In *Advances in Neural Information Processing Systems*, pages 12317–12328, 2019.

[12] 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.

[13] 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.

[14] 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.

[15] Yifan Ding, Liqiang Wang, Huan Zhang, Jinfeng Yi, Deliang Fan, and Boqing Gong. Defending against adversarial attacks using random forest. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[16] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[17] Justin Gilmer and Dan Hendrycks. A discussion of 'adversarial examples are not bugs, they are features': Adversarial example researchers need to expand what is meant by 'robustness'. *Distill*, 2019. doi: 10.23915/distill.00019.1.

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

[19] Moritz Hardt, Eric Price, and Nati Srebro. Equality of opportunity in supervised learning. In *Advances in neural information processing systems*, pages 3315–3323, 2016.

[20] Faisal Kamiran and Toon Calders. Classifying without discriminating. In *2009 2nd International Conference on Computer, Control and Communication*, pages 1–6. IEEE, 2009.

[21] Faisal Kamiran, Toon Calders, and Mykola Pechenizkiy. Discrimination aware decision tree learning. In *2010 IEEE International Conference on Data Mining*, pages 869–874. IEEE, 2010.

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

[23] Ron Larson and Robert P. Hostetler. *Precalculus: a concise course*. Houghton Mifflin, 2007.

[24] Hyafil Laurent and Ronald L Rivest. Constructing optimal binary decision trees is np-complete. *Information processing letters*, 5(1):15–17, 1976.

[25] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

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

[27] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1765–1773, 2017.

[28] 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.

[29] 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.

[30] Nicolas Papernot, Patrick Drew McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *1st IEEE European Symposium on Security and Privacy, EURO S and P 2016*, pages 372–387. Institute of Electrical and Electronics Engineers Inc., 2016.

[31] 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*, 12:2825–2830, 2011.

[32] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[33] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[34] Edward Raff, Jared Sylvester, and Steven Mills. Fair forests: Regularized tree induction to minimize model bias. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, pages 243–250, 2018.

[35] 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. ISSN 1012-2443. doi: 10.1023/B:AMAI.0000018580.96245.c6.

[36] 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.

[37] Sicco Verwer and Yingqian Zhang. Learning decision trees with flexible constraints and objectives using integer optimization. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 94–103. Springer, 2017.

[38] Christina Wadsworth, Francesca Vera, and Chris Piech. Achieving fairness through adversarial learning: an application to recidivism prediction. *arXiv preprint arXiv:1807.00199*, 2018.

[39] Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. Learning fair representations. In *International Conference on Machine Learning*, pages 325–333, 2013.

[40] Fuyong Zhang, Yi Wang, Shigang Liu, and Hua Wang. Decision-based evasion attacks on tree ensemble classifiers. *World Wide Web*, pages 1–21, 2020.