Provably Robust Boosted Decision Stumps and Trees against Adversarial Attacks

Maksym Andriushchenko
University of Tübingen
maksym.andriushchenko@uni-tuebingen.de

Matthias Hein
University of Tübingen
matthias.hein@uni-tuebingen.de

Abstract

The problem of adversarial robustness has been studied extensively for neural networks. However, for boosted decision trees and decision stumps there are almost no results, even though they are widely used in practice (e.g. XGBoost) due to their accuracy, interpretability, and efficiency. We show in this paper that for boosted decision stumps the *exact* min-max robust loss and test error for an l_∞-attack can be computed in $O(T \log T)$ time per input, where T is the number of decision stumps and the optimal update step of the ensemble can be done in $O(n^2\, T \log T)$, where n is the number of data points. For boosted trees we show how to efficiently calculate and optimize an upper bound on the robust loss, which leads to state-of-the-art robust test error for boosted trees on MNIST (12.5% for $\epsilon_\infty = 0.3$), FMNIST (23.2% for $\epsilon_\infty = 0.1$), and CIFAR-10 (74.7% for $\epsilon_\infty = 8/255$). Moreover, the robust test error rates we achieve are competitive to the ones of provably robust convolutional networks. The code of all our experiments is available at `http://github.com/max-andr/provably-robust-boosting`.

1 Introduction

It has recently been shown that deep neural networks are easily fooled by imperceptible perturbations called *adversarial examples* [62, 24] or tend to output high-confidence predictions on out-of-distribution inputs [51, 49, 29] that have nothing to do with the original classes. The most popular defense against adversarial examples is adversarial training [24, 45], which is formulated as a robust optimization problem [59, 45]. However, the inner maximization problem is likely to be NP-hard for neural networks as computing optimal adversarial examples is NP-hard [33, 71]. A large variety of sophisticated defenses proposed for neural networks [31, 7, 43] could be broken again via more sophisticated attacks [1, 18, 48]. Moreover, empirical robustness, evaluated by *some* attack, can also arise from gradient masking or obfuscation [1] in which case gradient-free or black-box attacks often break heuristic defenses. A solution to this problem are methods that lead to *provable robustness guarantees* [28, 72, 54, 77, 68, 75, 13, 25] or lead to classifiers which can be certified via exact combinatorial solvers [63]. However, these solvers do not scale to large neural networks, and networks having robustness guarantees lack in terms of prediction performance compared to standard ones. The only scalable certification method is randomized smoothing [41, 42, 12, 57], however obtaining tight certificates for norms other than l_2 is an open research question.

While the adversarial problem has been studied extensively for neural networks, other classifiers have received much less attention e.g. kernel machines [76, 56, 28], k-nearest neighbors [69], and decision trees [52, 3, 9]. Boosting, in particular boosted decision trees, are very popular in practice due to their interpretability, competitive prediction performance, and efficient recent implementations such as XGBoost [10] and LightGBM [34]. Thus there is also a need to develop boosting methods which are robust to worst-case measurement error or adversarial changes of the input data. While robust boosting has been extensively considered in the literature [70, 44, 19], it refers in that context to a

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

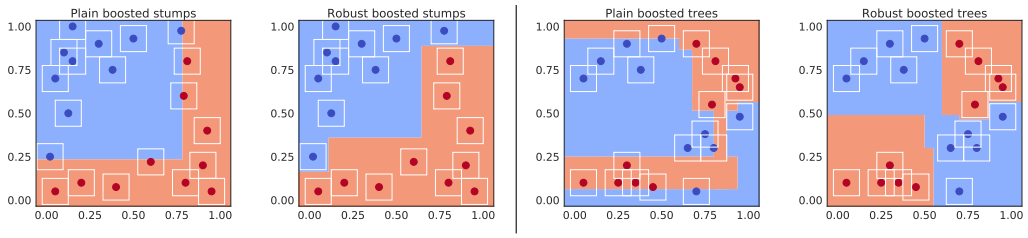

Figure 1: Left: boosted decision *stumps*: normal and our robust models. **Right**: boosted decision *trees*: normal and our robust models. In both cases, the normal models have very small geometric margin, while our robust models also classify all training points correctly but additionally enforce a large geometric margin.

large functional margin or robustness with respect to outliers e.g. via using a robust loss function, but not to the adversarial robustness we are considering in this paper. In the context of *adversarial* robustness, very recently [9] considered the robust min-max loss for an ensemble of decision trees with coordinate-aligned splits. They proposed an approximation of the inner maximization problem but without any guarantees. The robustness guarantees were then obtained via a mixed-integer formulation of [32] for the computation of the minimal adversarial perturbation for tree ensembles. However, this approach has limited scalability to large problems.

Contributions In this paper, we show how to exactly compute the robust loss and robust test error with respect to l_∞-norm perturbations for an ensemble of decision stumps with coordinate-aligned splits. This can be done efficiently in $O(T \log T)$ time per data point, where T is the number of decision stumps. Moreover, we show how to perform the globally optimal update of an ensemble of decision stumps by directly minimizing the robust loss without any approximation in $O(n^2\, T \log T)$ time per coordinate, where n is the number of training examples. We also derive a strict upper bound on the robust loss for tree ensembles based on our results for an ensemble of decision stumps. It can be efficiently evaluated in $O(T\, l)$ time, where l is the number of leaves in the tree. Then we show how this upper bound can be minimized during training in $O(n^2\, l)$ time per coordinate. Our derived upper bound is quite tight empirically and leads to provable guarantees on the robustness of the resulting tree ensemble. The difference of the resulting robust boosted decision stumps and trees compared to normally trained models is visualized in Figure 1.

2 Boosting and Robust Optimization for Adversarial Robustness

In this section we fix the notation, the framework of boosting, and define briefly the basis of robust optimization for adversarial robustness, underlying adversarial training. In the next sections we derive the specific robust training procedure for an ensemble of decision stumps where we optimize the exact robust loss and for a tree ensemble where we optimize an upper bound.

Boosting While the main ideas can be generalized to the multi-class setting (using one-vs-all, see Appendix E), for simplicity of the derivations we restrict ourselves to binary classification, that is our labels y are in $\{-1, 1\}$ and we assume to have d real-valued features. Boosting can be described as the task of fitting an ensemble $F : \mathbb{R}^d \to \mathbb{R}$ of weak learners $f_t : \mathbb{R}^d \to \mathbb{R}$ given as $F(x) = \sum_{t=1}^{T} f_t(x)$. The final classification is done via the sign of $F(x)$. In boosting the ensemble is fitted in a greedy way in the sense that given the already estimated ensemble we determine an update $F' = F + f_{T+1}$, by fitting the new weak learner f_{T+1} being guided by the performance of the current ensemble F. In this paper we use in the experiments the exponential loss $L : \mathbb{R} \to \mathbb{R}$, where we use the functional margin formulation where for a point $(x, y) \in \mathbb{R}^d \times \{-1, 1\}$ it is defined as $L(y\, f(x)) = \exp(-y\, f(x))$. However, all following algorithms and derivations hold for any margin-based, strictly monotonically decreasing, convex loss function L, e.g. logistic loss $L(y\, f(x)) = \ln(1 + \exp(-yf(x)))$. The advantage of the exponential loss is that it decouples F and the update f_{T+1} in the estimation process and allows us to see the estimation process for f_{T+1} as fitting a weighted exponential loss where the weights to fit (x, y) are given by $\exp(-y\, F(x))$,

$$L(y\, F'(x)) = \exp\big(-y\,\big(F(x) + f_{T+1}(x)\big)\big) = \exp\big(-y\, F(x)\big)\exp\big(-y\, f_{T+1}(x)\big).$$

In this paper we consider as weak learners: a) decision stumps (i.e. trees of depth one) of the form $f_{t,i} : \mathbb{R}^d \to \mathbb{R}$, $f_{t,i}(x) = w_l + w_r \mathbb{1}_{x_i \geq b}$, where one does a coordinate-aligned split and b) decision

trees (binary tree) of the form $f_t(x) = u^{(t)}_{q_t(x)}$, where $u^{(t)}_{q_t(x)} : V \to \mathbb{R}$ is a mapping from the set of leaves V of the tree to $\mathbb{R}$ and $q_t : \mathbb{R}^d \to V$ is a mapping which assigns to every input the leaf of the tree it ends up. While the approach can be generalized to general linear splits of the form, $w_l + w_r \mathbb{1}_{\langle v,x\rangle \geq b}$, we concentrate on coordinate-aligned splits, $w_l + w_r \mathbb{1}_{x_i \geq b}$ which are more common in practice since they lead to competitive performance and are easier to interpret for humans.

Robust optimization for adversarial robustness Finding the minimal perturbation with respect to some l_p-distance can be formulated as the following optimization problem:

$$\min_{\delta \in \mathbb{R}^d} \|\delta\|_p \quad \text{such that} \quad y_i f(x_i + \delta) \leq 0, \quad x_i + \delta \in C \tag{1}$$

where $(x_i, y_i) \in \mathbb{R}^d \times \{-1, 1\}$ and C is a set of constraints every input has to fulfill. In this paper we assume $C = [0, 1]^d$ and that all features are normalized to be in this range. We emphasize that we concentrate on *continuous* features, for adversarial perturbations of *discrete* features we refer to [53, 17, 36]. We denote by δ_i^* the optimal solution of this problem for (x_i, y_i). Furthermore, let $\Delta_p(\epsilon) := \{\delta \in \mathbb{R}^d \mid \|\delta\|_p \leq \epsilon\}$ be the set of perturbations with respect to which we aim to be robust. Then the *robust test error* with respect to $\Delta_p(\epsilon)$ is defined for n data points as $\frac{1}{n}\sum_{i=1}^n \mathbb{1}_{\|\delta_i^*\|_p \leq \epsilon}$.

The optimization problem (1) is non-convex for neural networks and can only be solved exactly via mixed-integer programming [63] which scales exponentially with the number of hidden neurons. Since such an evaluation is prohibitively expensive in most cases, often robustness is evaluated via heuristic attacks [47, 45, 8] which results in lower bounds on the robust test error. Provable robustness aims at providing upper bounds on the robust test error and the optimization of these bounds during training [28, 72, 54, 77, 75, 13, 25, 12]. For an ensemble of trees the optimization problem (1) can also be reformulated as a mixed-integer-program [32] which does not scale to large ensembles.

The goal of improving adversarial robustness can be formulated as a robust optimization problem with respect to the set of allowed perturbations $\Delta_p(\epsilon)$ [59, 45]:

$$\min_\theta \sum_{i=1}^n \max_{\delta \in \Delta_p(\epsilon)} L\big(f(x_i + \delta; \theta), y_i\big). \tag{2}$$

A training process, where one tries at each update step to approximately solve the inner maximization problem, is called *adversarial training* [24]. We note that the maximization problem is in general non-concave and thus globally optimal solutions are very difficult to obtain. Our goal in the following two sections is to get provable robustness guarantees for boosted stumps and trees by directly optimizing (2) or an upper bound on the inner maximization problem.

3 Exact Robust Optimization for Boosted Decision Stumps

We first show how the exact robust loss $\max_{\delta \in \Delta_p(\epsilon)} L(y_i\, F(x_i + \delta; \theta))$ can be computed for an ensemble F of decision stumps. While decision stumps are very simple weak learners, they have been used in the original AdaBoost [20] and were successfully used in object detection [66] or face detection [67] which could be done in real-time due to the simplicity of the classifier.

3.1 Exact Robust Test Error for Boosted Decision Stumps

The ensemble of decision stumps can be written as

$$F(x) = \sum_{t=1}^T f_{t,c_t}(x) = \sum_{t=1}^T \Big(w_l^{(t)} + w_r^{(t)} \mathbb{1}_{x_{c_t} \geq b_t}\Big),$$

where c_t is the coordinate for which f_t makes a split. First, observe that a point $x \in \mathbb{R}^d$ with label y is correctly classified when $yF(x) > 0$. In order to determine whether the point x is adversarially robust wrt l_∞-perturbations, one has to solve the following optimization problem:

$$G(x, y) := \min_{\|\delta\|_\infty \leq \epsilon} yF(x + \delta) \tag{3}$$

If $G(x, y) \leq 0$, then the point x is non-robust. If $G(x, y) > 0$, then the point x is robust, i.e. it is not possible to change the class. Thus the exact minimization of (3) over the test set yields the exact robust test error. For many state-of-the-art classifiers, this problem is NP-hard. For particular MIP formulations for tree ensembles, see [32], or for neural networks, see [63]. Closed-form solutions are known only for the simplest models such as linear classifiers [24].

We can solve this certification problem for the robust test error exactly and efficiently by noting that the objective and the attack model $\Delta_\infty(\epsilon)$ is *separable* wrt the input dimensions. Therefore, we have to solve up to d simple *one-dimensional* optimization problems. We denote $S_k = \{s \in \{1, \dots, T\} \mid c_s = k\}$, i.e. the set of stump indices that split coordinate k. Then

$$\min_{\|\delta\|_\infty \leq \epsilon} yF(x+\delta) = \min_{\|\delta\|_\infty \leq \epsilon} \sum_{t=1}^{T} y f_{t,c_t}(x+\delta) = \min_{\|\delta\|_\infty \leq \epsilon} \sum_{k=1}^{d} \sum_{s \in S_k} y f_{s,k}(x+\delta) \tag{4}$$

$$= \sum_{k=1}^{d} \min_{|\delta_k| \leq \epsilon} \sum_{s \in S_k} y f_{s,k}(x+\delta) = \sum_{k=1}^{d} \Big[\sum_{s \in S_k} y w_l^{(s)} + \min_{|\delta_k| \leq \epsilon} \sum_{s \in S_k} y w_r^{(s)} \mathbb{1}_{x_k + \delta_k \geq b_s} \Big] \coloneqq \sum_{k=1}^{d} G_k(x, y)$$

The one-dimensional optimization problem $\min_{|\delta_k| \leq \epsilon} \sum_{s \in S_k} y w_r^{(s)} \mathbb{1}_{x_k + \delta_k \geq b_s}$ can be solved by simply checking all $|S_k| + 1$ piece-wise constant regions of the classifier for $\delta_k \in [-\epsilon, \epsilon]$. The detailed algorithm can be found in Appendix B. The overall time complexity of the exact certification is $O(T \log T)$ since we need to sort up to T thresholds b_s in ascending order to efficiently calculate all partial sums of the objective. Moreover, using this result, we can obtain provably minimal adversarial examples (see Appendix B for details and Figure 11 for visualizations).

3.2 Exact Robust Loss Minimization for Boosted Decision Stumps

We note that when L is monotonically decreasing, it holds:

$$\max_{\delta \in \Delta_\infty(\epsilon)} L(y\, F(x+\delta)) = L\Big(\min_{\delta \in \Delta_\infty(\epsilon)} yF(x+\delta) \Big),$$

and thus the certification algorithm can directly be used to compute also the robust loss. For updating the ensemble F with a new stump f that splits a certain coordinate j, we first have to solve the inner maximization problem over $\Delta_\infty(\epsilon)$ in (2) before[1] we optimize the parameters w_l, w_r, b of f:

$$\max_{\|\delta\|_\infty \leq \epsilon} L\Big(y_i F(x_i+\delta) + y_i f_j(x_i+\delta) \Big) = L\Big(\min_{\|\delta\|_\infty \leq \epsilon} \big[\sum_{k=1}^{d} \sum_{s \in S_k} y_i f_{s,k}(x_i+\delta) + y_i f_j(x_i+\delta) \big] \Big)$$

$$= L\Big(\sum_{k \neq j} \min_{|\delta_k| \leq \epsilon} \sum_{s \in S_k} y_i f_{s,k}(x_i+\delta) + \min_{|\delta_j| \leq \epsilon} \big[\sum_{s \in S_j} y_i f_{s,j}(x_i+\delta) + y_i f_j(x_i+\delta) \big] \Big)$$

$$= L\Big(\sum_{k \neq j} G_k(x_i, y_i) + \sum_{s \in S_j} y_i w_l^{(s)} + y_i w_l + \min_{|\delta_j| \leq \epsilon} \big[\sum_{s \in S_j} y_i w_r^{(s)} \mathbb{1}_{x_{ij} + \delta_j \geq b_s} + y_i w_r \mathbb{1}_{x_{ij} + \delta_j \geq b} \big] \Big).$$

In order to solve the remaining optimization problem for δ_j we have to make a case distinction based on the values of w_r. However, first we define the minimal values of the ensemble part on $\delta_j \in [-\epsilon, b - x_{ij})$ and $\delta_j \in [b - x_{ij}, \epsilon]$ as

$$h_l(x_{ij}, y_i) \coloneqq \min_{\substack{\delta_j < b - x_{ij} \\ |\delta_j| \leq \epsilon}} \sum_{s \in S_j} y_i w_r^{(s)} \mathbb{1}_{x_{ij} + \delta_j \geq b_s}, \quad h_r(x_{ij}, y_i) \coloneqq \min_{\substack{\delta_j \geq b - x_{ij} \\ |\delta_j| \leq \epsilon}} \sum_{s \in S_j} y_i w_r^{(s)} \mathbb{1}_{x_{ij} + \delta_j \geq b_s}$$

These problems can be solved analogously to $G_k(x, y)$. Then we get the case distinction:

$$g(x_{ij}, y_i; w_r) = \min_{|\delta_j| \leq \epsilon} \big[\sum_{s \in S_j} y_i w_r^{(s)} \mathbb{1}_{x_{ij} + \delta_j \geq b_s} + y_i w_r \mathbb{1}_{x_{ij} + \delta_j \geq b} \big] \tag{5}$$

$$= \begin{cases} h_r(x_{ij}, y_i) + y_i w_r & \text{if } b - x_{ij} < -\epsilon \text{ or } (|b - x_{ij}| \leq \epsilon \text{ and } h_l(x_{ij}, y_i) > h_r(x_{ij}, y_i) + y_i w_r) \\ h_l(x_{ij}, y_i) & \text{if } b - x_{ij} > \epsilon \quad \text{or } (|b - x_{ij}| \leq \epsilon \text{ and } h_l(x_{ij}, y_i) \leq h_r(x_{ij}, y_i) + y_i w_r) \end{cases}$$

The following Lemma shows that the robust loss is jointly convex in w_l, w_r $\big($to see this set $l = 2$, $u = (w_l, w_r)^T$, $r(\hat{x}) = (y_i, y_i \mathbb{1}_{\hat{x}_{ij} \geq b})^T$, $C = B_\infty(x_i, \epsilon)$ and $c = \sum_{k \neq j} G_k(x_i, y_i)\big)$.

[1]The order is very important as a min-max problem is not the same as a max-min problem.

Lemma 1 *Let $L : \mathbb{R} \to \mathbb{R}$ be a convex, monotonically decreasing function. Then $\tilde{L} : \mathbb{R}^l \to \mathbb{R}$ defined as $\tilde{L}(u) = \max_{\tilde{x} \in C} L(c + \langle r(\tilde{x}), u \rangle)$ is convex for any $c \in \mathbb{R}$, $r : \mathbb{R}^d \to \mathbb{R}^l$, and $C \subseteq \mathbb{R}^d$.*

Thus the loss term for each data point is jointly convex in w_l, w_r and consequently the sum of the losses is convex as well. This means that for the overall robust optimization problem over the parameters w_l, w_r (for a fixed b), we have to minimize the following convex function

$$L^*(j, b) = \min_{w_l, w_r} \sum_{i=1}^{n} L\Big(\sum_{k \neq j} G_k(x_i, y_i) + \sum_{s \in S_j} y_i w_l^{(s)} + y_i w_l + g(x_{ij}, y_i; w_r) \Big).$$

We plot an example of this objective wrt the parameters w_l and w_r of a single decision stump in Figure 2. In general, for an arbitrary loss L, there is no closed-form minimizer wrt w_l and w_r. Thus, we can minimize such an objective using, e.g. coordinate descent. Then on every iteration of coordinate descent the minimum wrt w_l or w_r can be found using bisection for any convex loss L. For the exponential loss, we can optimize wrt w_l via a closed-form minimizer when w_r is fixed. The details can be found in Appendix B.3.

Finally, we have to minimize over all possible thresholds. We choose the potential thresholds $b \in B_j = \{x_{ij} - \epsilon - \nu, x_{ij} + \epsilon + \nu \mid i = 1, \dots, n\}$, where ν can be as small as precision allows and is just introduced so that the thresholds lie outside of $\Delta_\infty(\epsilon)$. We optimize the robust loss $L^*(j, b)$ for all thresholds $b \in B_j$ and determine the minimum. For each contiguous set of minimizers we determine the nearest neighbors in B_j and check the thresholds half-way to them (note that they have at most the same robust loss but never a better one) and then take the threshold in the middle of all the ones having equal loss. As there are in the worst case $2n$ unique thresholds, the overall complexity of one update step is $O(n^2 \, T \log T)$. And finally, at each update step one typically checks all d coordinates and takes the one which yields the smallest overall robust loss of the ensemble.

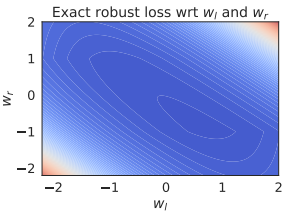

Figure 2: Visualization of the min-max objective which is convex wrt the parameters w_l and w_r of a decision stump.

4 Robust Optimization for Boosted Decision Trees

We first provide an upper bound on the robust test error of the tree ensemble which is used further to derive an upper bound on the robust loss that is then minimized in the update step of tree ensemble.

4.1 Upper Bound on the Robust Test Error for Boosted Decision Trees

Our goal is to solve the optimization problem (3). While the exact minimization is NP-hard for tree ensembles [32], we can similarly to [73, 54] for neural networks derive a tractable lower bound $\tilde{G}(x, y)$ on $G(x, y)$ for an ensemble of trees:

$$\min_{\|\delta\|_p \leq \epsilon} yF(x + \delta) = \min_{\|\delta\|_p \leq \epsilon} \sum_{t=1}^{T} y u^{(t)}_{q_t(x+\delta)} \geq \sum_{t=1}^{T} \min_{\|\delta\|_p \leq \epsilon} y u^{(t)}_{q_t(x+\delta)} \coloneqq \tilde{G}(x, y) \tag{6}$$

If $\tilde{G}(x, y) > 0$, then the point x is provably robust. However, if $\tilde{G}(x, y) \leq 0$, the point may be either robust or non-robust. In this way, we get an upper bound on the number of non-robust points, which yields an *upper bound* on the robust test error. We note that for a decision tree, $\min_{\|\delta\|_p \leq \epsilon} y u^{(t)}_{q_t(x+\delta)}$ can be found exactly by checking all leafs which are reachable for points in $B_p(x, \epsilon)$. This can be done in $O(l)$ time per tree, where l is the number of leaves in the tree.

4.2 Minimization of an Upper Bound on the Robust Loss for Boosted Decision Trees

The goal is to upper bound the inner maximization problem of Equation (2) based on the certificate that we derived. Note that we aim to *bound* the loss of the whole ensemble $F + f$, and thus we do not use any approximations of the loss such as the second-order Taylor expansion used in [23, 10]. We use $p = \infty$, that is the attack model is $\Delta_\infty(\epsilon)$. Let $F(x) = \sum_{t=1}^{T} f_t(x) = \sum_{t=1}^{T} u^{(t)}_{q_t(x)}$ be a fixed

ensemble of trees and f a new tree with which we update the ensemble. Then the robust optimization problem is:

$$\min_{f} \sum_{i=1}^{n} \max_{\|\delta\|_\infty \leq \epsilon} L\Big(y_i\big(F(x_i+\delta)+f(x_i+\delta)\big)\Big) \tag{7}$$

The inner maximization problem can be upper bounded for every tree separately given that $L(yf(x))$ is monotonically decreasing wrt $yf(x)$, and using our certificate for the ensemble of $T+1$ trees:

$$\max_{\|\delta\|_\infty \leq \epsilon} L\Big(y_i F(x_i+\delta)+y_i f(x_i+\delta)\Big) = L\Big(\min_{\|\delta\|_\infty \leq \epsilon}\Big[\sum_{t=1}^{T} y_i f_t(x_i+\delta)+y_i f(x_i+\delta)\Big]\Big) \tag{8}$$

$$\leq L\Big(\sum_{t=1}^{T}\min_{\|\delta\|_\infty \leq \epsilon} y_i f_t(x_i+\delta)+\min_{\|\delta\|_\infty \leq \epsilon} y_i f(x_i+\delta)\Big) = L\Big(\tilde{G}(x_i,y_i)+\min_{\|\delta\|_\infty \leq \epsilon} y_i f(x_i+\delta)\Big)$$

We can efficiently calculate $\tilde{G}(x_i, y_i)$ as described in the previous subsection. But note that $\min_{\|\delta\|_\infty \leq \epsilon} y_i f(x_i+\delta)$ depends on the tree f. The exact tree fitting is known to be NP-complete [39], although it is still possible to scale it to some moderate-sized problems with recent advances in MIP-solvers and hardware as shown in [2]. We want to keep the overall procedure scalable to large datasets, so we will stick to the standard greedy recursive algorithm for fitting the tree. On every step of this process, we fit for some coordinate $j \in \{1,\dots,d\}$ and for some splitting threshold b, a single decision stump $f(x) = w_l + w_r \mathbb{1}_{x_j \geq b}$. Therefore, for a particular decision stump with threshold b and coordinate j we have to solve the following problem:

$$\min_{w_l, w_r \in \mathbb{R}} \sum_{i \in I} L\left(\tilde{G}(x_i,y_i)+y_i w_l+\min_{|\delta_j| \leq \epsilon} y_i w_r \mathbb{1}_{x_{ij}+\delta_j \geq b}\right) \tag{9}$$

where I are all the points $x_i+\delta$ which can reach this leaf for some δ with $\|\delta\|_\infty \leq \epsilon$.

Finally, we have to make a case distinction depending on the values of w_r and $b - x_{ij}$:

$$\min_{|\delta_j| \leq \epsilon} y_i w_r \mathbb{1}_{x_{ij}+\delta_j \geq b} = y_i w_r \cdot \begin{cases} 1 & \text{if } b - x_{ij} < -\epsilon \text{ or } (|b-x_{ij}| \leq \epsilon \text{ and } y_i w_r < 0) \\ 0 & \text{if } b - x_{ij} > \epsilon \quad \text{or } (|b-x_{ij}| \leq \epsilon \text{ and } y_i w_r \geq 0) \end{cases} \tag{10}$$

where we denote the case distinction for brevity as $\mathbb{1}(x_i, y_i; w_r)$. Note that the right side of (10) is concave as a function of w_r. Thus the overall robust optimization amounts to finding the minimum of the following objective, which is again by Lemma 1 jointly convex in w_l, w_r:

$$L^*(j,b) = \min_{w_l, w_r} \sum_{i:i \in I} L\left(\tilde{G}(x_i,y_i)+y_i w_l+y_i w_r \mathbb{1}(x_i,y_i;w_r)\right) \tag{11}$$

Note that the case distinction $\mathbb{1}(x_i, y_i; w_r)$ can be fixed once we fix the sign of w_r. This allows us to avoid doing bisection on w_r, and rather use coordinate descent directly on each interval $w_r \geq 0$ and $w_r < 0$. After finding the minimum of the objective on each interval, we then combine the results from both intervals by taking the smallest loss out of them. The details are given in Appendix B.3.

Then we select the optimal threshold as described in Section 3.2. Finally, as in other tree building methods such as [5, 10], we perform pruning after a tree is constructed. We start from the leafs and prune nodes based on the upper bound on the training robust loss (8) to ensure that it decreases at every iteration of tree boosting. This cannot be guaranteed with robust splits without pruning since the tree construction process is greedy, and some training examples are also influenced by splits at different branches. Thus, in order to control the upper bound on the robust loss globally over the whole tree as in (8), and not just for the current subtree as in (9), we need a post-hoc approach that takes into account the structure of the whole tree. Therefore, we have to use pruning. We note that in the extreme case, pruning may leave only one decision stump at the root (although it happens extremely rarely in practice), for which we are guaranteed to decrease the upper bound on the robust loss. Thus every new tree in the ensemble is guaranteed to reduce the upper bound on the robust loss. Note that this is also true if we use the shrinkage parameter [21] which we discuss in Appendix C.

Lastly, we note that the total worst case complexity is $O(n^2)$ in the number of training examples compared to $O(n \log n)$ for XGBoost, which is a relatively low price given that the overall optimization problem is significantly more complicated than the formulation used in XGBoost.

5 Experiments

General setup We are primarily interested in two quantities: test error (TE) and robust test error (RTE) wrt l_∞-perturbations. For boosted stumps, we compute RTE as described in Section 3.1, but we also report the upper bound on RTE (URTE) obtained using the stump-wise bound from Section 4.1 to illustrate that it is actually tight for almost all models. For boosted trees, we report RTE obtained via the MIP formulation of [32] which we adapted to a feasibility problem (see Appendix G.2 for more details), and also the tree-wise upper bounds described in Section 4.1. For evaluation we use 11 datasets: breast-cancer, diabetes, cod-rna, MNIST 1-5 (digit 1 vs digit 5), MNIST 2-6 (digit 2 vs digit 6, following [32] and [9]), FMNIST shoes (sandals vs sneakers), GTS 100-rw (speed 100 vs roadworks), GTS 30-70 (speed 30 vs speed 70), MNIST, FMNIST, and CIFAR-10. More details about the datasets are given in Appendix F. We emphasize that we evaluate our models on image recognition datasets mainly for the sake of comparison to other methods reported in the literature.

We consider five types of boosted stumps: normally trained stumps, adversarially trained stumps (see Appendix G.1 for these results), robust stumps of Chen et al. [9], our robust stumps where the robust loss is bounded stump-wise, and our robust stumps where the robust loss is calculated exactly. Next we consider four types of boosted trees: normally trained trees, adversarially trained trees, robust trees of Chen et al. [9], and our robust trees where the robust loss is bounded tree-wise. Both for stumps and trees, we perform l_∞ adversarial training following [32], i.e. every iteration we train on clean training points and adversarial examples (equal proportion). We generate adversarial examples via the *cube attack* – a simple attack inspired by random search [50] described in Appendix D (we use 10 iterations and $p = 0.5$) and its performance is shown in Section G.3. We perform model selection of our models and models from Chen et al. [9] based on the validation set of 20% randomly selected points from the original training set, and we train on the rest of the training set. All models are trained with the exponential loss. More details about the experiments are available in Appendix F and in our repository `http://github.com/max-andr/provably-robust-boosting`.

Boosted decision stumps The results for boosted stumps are given in Table 1. First, we observe that normal models are not robust for the considered l_∞-perturbations. However, both variants of our robust boosted stumps significantly improve RTE, outperforming the method of Chen et al. [9] on 7 out of the 8 datasets. Note that although our exact method optimizes the exact robust loss, we are still not guaranteed to *always* outperform Chen et al. [9] since they use a different loss function, and the quantities of interest are calculated on test data. The largest improvements compared to normal models are obtained on breast-cancer from 98.5% RTE to 10.9% and on MNIST 2-6 from 99.9% to 9.1% RTE. The robust models perform slightly worse in terms of test error, which is in line with the empirical observation made for adversarial training for neural networks [64]. Additionally, to the robust test error (RTE), we also report the upper bound (URTE) to show that it is very close to RTE. Notably, for our robust stumps trained with the upper bound on the robust loss, URTE is equal to the RTE for all models, and it is very close to the RTE of our robust stumps trained with the exact robust loss, while taking about 4x less time to train in average. Thus bounding the sum over weak learners element-wise, as done in (6), seems to be tight enough to yield robust models. Finally, we provide in Appendix G.2 a more detailed comparison to the robust boosted stumps of Chen et al. [9].

Table 1: Evaluation of robustness for boosted stumps. We show, in percentage, test error (TE), exact robust test error (RTE), and upper bound on robust test error (URTE). Both variants of our robust stumps outperform the method of Chen et al. [9]. We also observe that URTE is very close to RTE or even the same for many models.

		Normal stumps (standard training)			Robust stumps Chen et al. [9]		Our robust stumps (robust loss bound)			Our robust stumps (exact robust loss)		
Dataset	l_∞ ϵ	TE	RTE	URTE	TE	RTE	TE	RTE	URTE	TE	RTE	URTE
breast-cancer	0.3	**2.9**	98.5	100	8.8	16.8	4.4	**10.9**	10.9	5.1	**10.9**	10.9
diabetes	0.05	24.7	54.5	56.5	**23.4**	**30.5**	28.6	33.1	33.1	27.3	31.8	31.8
cod-rna	0.025	**4.7**	42.8	44.9	11.6	23.2	11.2	**22.4**	22.4	11.2	22.6	22.6
MNIST 1-5	0.3	**0.5**	85.4	85.4	0.9	5.2	0.6	3.7	3.7	0.7	**3.6**	3.7
MNIST 2-6	0.3	**1.7**	99.9	99.9	2.8	13.9	3.0	**9.1**	9.1	3.0	9.2	9.2
FMNIST shoes	0.1	**2.4**	100	100	7.1	22.2	6.2	11.8	11.8	5.7	**10.8**	11.5
GTS 100-rw	8/255	**1.1**	9.9	9.9	2.0	11.8	2.8	8.9	8.9	2.0	**6.7**	6.7
GTS 30-70	8/255	**11.3**	53.7	53.7	12.7	28.2	12.7	**26.9**	26.9	12.9	27.6	27.6

Table 2: Evaluation of robustness for boosted *trees*. We report, in percentages, test error (TE), robust test error (RTE), and upper bound on robust test error (URTE). Our robust boosted trees lead to better RTE compared to adversarial training and robust trees of Chen et al. [9]. We observe that especially for our models URTE are very close to RTE, while URTE are orders of magnitude faster to compute.

		Normal trees (standard training)			Adv. trained trees (with cube attack)			Robust trees Chen et al. [9]		Our robust trees (robust loss bound)		
Dataset	l_∞ ϵ	TE	RTE	URTE	TE	RTE	URTE	TE	RTE	TE	RTE	URTE
breast-cancer	0.3	0.7	81.0	81.8	**0.0**	27.0	27.0	0.7	13.1	0.7	**6.6**	6.6
diabetes	0.05	22.7	55.2	61.7	26.6	46.8	46.8	**22.1**	40.3	27.3	**35.7**	35.7
cod-rna	0.025	**3.4**	37.6	47.1	10.9	24.8	24.8	10.2	24.2	6.9	**21.3**	21.4
MNIST 1-5	0.3	**0.1**	90.7	96.0	1.3	9.0	9.5	0.3	2.9	0.2	**1.3**	1.4
MNIST 2-6	0.3	**0.4**	89.6	100	2.3	15.1	15.9	0.5	6.9	0.7	**3.8**	4.1
FMNIST shoes	0.1	**1.7**	99.8	99.9	5.5	14.1	14.2	3.1	13.2	3.6	**8.0**	8.1
GTS 100-rw	8/255	**0.9**	6.0	6.1	1.0	8.4	8.4	1.5	9.7	2.6	**4.7**	4.7
GTS 30-70	8/255	14.2	31.4	32.6	16.2	26.7	26.8	**11.5**	28.8	13.8	**20.9**	21.4

Boosted decision trees The results for boosted trees of depth 4 are given in Table 2. Our robust training of boosted trees outperforms both adversarial training and the method of Chen et al. [9] in terms of RTE on all 8 datasets. For example, on breast-cancer, the RTE of the robust trees of Chen et al. [9] is 13.1%, while the RTE of our robust model is 6.6% and we achieve the same test error of 0.7%. We note that TE and RTE of our robust trees are in many cases better than for our robust stumps. This suggests that there is a benefit of using more expressive weak learners in boosting to get more robust and accurate models. Adversarial training performs worse than our provable defense not only in terms of URTE, but even in terms of LRTE. This is different from the neural network literature [45, 25], where adversarial training usually provides better LRTE and significantly better test error than methods providing provable robustness guarantees. However, our upper bound on the robust loss is *tight* and *tractable* and thus adversarial training should not be used as it provides only a lower bound and minimization of an upper bound makes more sense than minimization of a lower bound. We provide a more detailed comparison to Chen et al. [9] in Appendix G.2 including multi-class datasets (MNIST, FMNIST). We also show there that our proposed method to calculate the certified robust error (URTE) is orders of magnitudes faster than the MIP formulation.

Comparison to provable defenses for neural networks We note that our methods are primarily suitable for tabular data, but in the literature on robustness of neural networks there are no established tabular datasets to compare to. Thus, we compare our robust boosted trees to the convolutional networks of [73, 16, 75, 25, 13] on MNIST, FMNIST, and CIFAR-10. We do not compare to randomized smoothing since it is competitive only for small l_∞-balls [57]. Since the considered datasets are multi-class, we extend our training of robust boosted trees from the binary classification case to multi-class using the one-vs-all approach described in Appendix E. We also use data augmentation by shifting the images by one pixel horizontally and vertically. We fit our robust trees with depth of up to 30 for MNIST and FMNIST, and with depth of up to 4 for CIFAR-10. Note that we restrict the minimum number of examples in a leaf to 100. Thus a tree of depth 30 makes only a small fraction of the possible 2^{30} splits. We provide a comparison in Table 3. In terms of provable robustness (URTE), our method is competitive to many provable defenses for CNNs. In particular, we outperform the LP-relaxation approach of [73] on all three datasets both in terms of test error and upper bounds. We also systematically outpeform the recent approach of [75] aiming at enhancing verifiability of CNNs – we have a better URTE with the same or better test error. Only the recent work of [25] is able to outperform our approach. Also, the CIFAR-10 model of [16] shows better URTE than our approach, but worse test error. We would like to emphasize that even on CIFAR-10 (with a relatively large $\epsilon = 8/255$) our models are not too far away from the state-of-the-art. In addition our robust boosted tree models require less computations at inference time.

Robustness vs accuracy tradeoff There is a lot of empirical evidence that robust training methods for neural networks exhibit a trade-off between robustness and accuracy [73, 25, 64]. We can confirm that the trade-off can also be observed for boosted trees: we consistently lose accuracy once we increase ϵ. The only *slight* gain in accuracy that we observe is on FMNIST shoes dataset. More details and plots of robustness versus accuracy can be found in Appendix G.4.

Table 3: Comparison of our robust boosted trees to the state-of-the-art provable defenses for convolutional neural networks reported in the literature. Our models are competitive to them in terms of upper bounds on robust test error (URTE). By * we denote results taken from [25] where they could achieve significantly better TE and URTE with the code of [73].

Dataset	l_∞ ϵ	Approach	TE	LRTE	URTE
MNIST	0.3	Wong et al. [73]*	13.52%	26.16%	26.92%
		Xiao et al. [75]	2.67%	7.95%	19.32%
		Our robust trees, depth 30	2.68%	12.46%	12.46%
		Gowal et al. [25]	1.66%	6.12%	**8.05%**
FMNIST	0.1	Wong and Kolter [72]	21.73%	31.63%	34.53%
		Croce et al. [13]	14.50%	26.60%	30.70%
		Our robust trees, depth 30	14.15%	23.17%	**23.17%**
CIFAR-10	8/255	Xiao et al. [75]	59.55%	73.22%	79.73%
		Wong et al. [73]	71.33%	–	78.22%
		Our robust trees, depth 4	58.46%	74.69%	74.69%
		Dvijotham et al. [16]	59.38%	67.68%	70.79%
		Gowal et al. [25]	50.51%	65.23%	**67.96%**

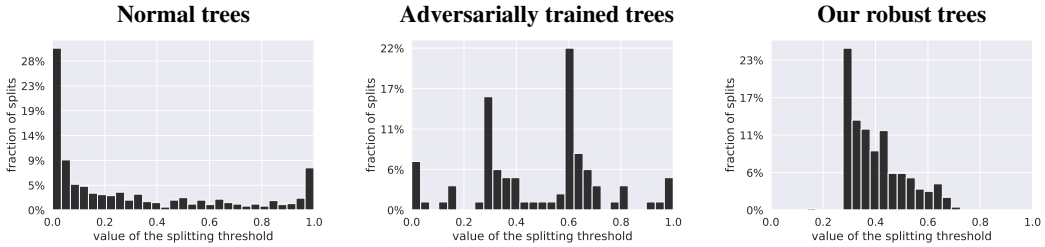

Figure 3: The distribution of the splitting thresholds for boosted trees models trained on MNIST 2-6. We can observe that our robust model almost always selects splits in the range between 0.3 and 0.7, which is reasonable given l_∞-perturbations within $\epsilon = 0.3$. At the same time, the normal and adversarially trained models split close to 0 or 1, which suggests that their decisions might be easily flipped by the adversary.

Interpretability For boosted stumps or trees, unlike for neural networks, we can *directly* inspect the model and the classification rules it has learned. In particular, in Figure 3, we plot the distibution of the splitting thresholds b for the three boosted trees models on MNIST 2-6 reported in Table 2. We can observe that our robust model almost always selects splits in the range between 0.3 and 0.7, which is reasonable given that more than 80% pixels of MNIST are either 0 or 1, and the considered l_∞-perturbations are within $\epsilon = 0.3$. At the same time, the normal and adversarially trained models split arbitrarily close to 0 or 1, which suggests that their decisions might be easily flipped if the adversary is allowed to change them within this ϵ. To emphasize the importance of interpretability and transparent decision making, we provide feature importance plots and more histograms of the splitting thresholds in Appendix G.5 and G.6.

6 Conclusions and Outlook

Our results show that the proposed methods achieve state-of-the-art provable robustness among boosted stumps and trees, and are also competitive to provably robust CNNs. This can be seen as a strong indicator that particularly for large l_∞-balls, current provably robust CNNs are so over-regularized that their performance is comparable to simple decision tree ensembles that make decisions based on individual pixel values. Thus it remains an open research question whether it is possible to establish tight and tractable upper bounds on the robust loss for neural networks. On the contrary, as shown in this paper, for boosted decision trees there exist simple and tight upper bounds which can be efficiently optimized. Moreover, for boosted decision stumps one can compute and optimize the exact robust loss. We thus think that if provable robustness is the goal then our robust decision stumps and trees are a promising alternative as they not only come with tight robustness guarantees but also are much easier to interpret.

Acknowledgements

We thank the anonymous reviewers for very helpful and thoughtful comments. We acknowledge the support from the German Federal Ministry of Education and Research (BMBF) through the Tübingen AI Center (FKZ: 01IS18039A). This work was also supported by the DFG Cluster of Excellence "Machine Learning – New Perspectives for Science", EXC 2064/1, project number 390727645, and by DFG grant 389792660 as part of TRR 248.

References

[1] Anish Athalye, Nicholas Carlini, and David A. Wagner. Obfuscated gradients give a false sense of security: circumventing defenses to adversarial examples. *ICML*, 2018.

[2] Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 2017.

[3] Dimitris Bertsimas, Jack Dunn, Colin Pawlowski, and Ying Daisy Zhuo. Robust classification. *INFORMS Journal on Optimization*, 1:2–34, 2018.

[4] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[5] Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone. Classification and regression trees. *Chapman & Hall/CRC*, 1984.

[6] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: reliable attacks against black-box machine learning models. *ICLR*, 2018.

[7] Jacob Buckman, Aurko Roy, Colin Raffel, and Ian Goodfellow. Thermometer encoding: one hot way to resist adversarial examples. *ICLR*, 2018.

[8] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. *IEEE Symposium on Security and Privacy*, 2017.

[9] Hongge Chen, Huan Zhang, Duane Boning, and Cho-Jui Hsieh. Robust decision trees against adversarial examples. *ICML*, 2019.

[10] Tianqi Chen and Carlos Guestrin. Xgboost: a scalable tree boosting system. *KDD*, 2016.

[11] Minhao Cheng, Thong Le, Pin-Yu Chen, Jinfeng Yi, Huan Zhang, and Cho-Jui Hsieh. Query-efficient hard-label black-box attack: an optimization-based approach. *ICLR*, 2019.

[12] Jeremy M Cohen, Elan Rosenfeld, and J Zico Kolter. Certified adversarial robustness via randomized smoothing. *ICML*, 2019.

[13] Francesco Croce, Maksym Andriushchenko, and Matthias Hein. Provable robustness of relu networks via maximization of linear regions. *AISTATS*, 2019.

[14] Stefan Droste, Thomas Jansen, and Ingo Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 2002.

[15] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

[16] Krishnamurthy Dvijotham, Sven Gowal, Robert Stanforth, Relja Arandjelovic, Brendan O'Donoghue, Jonathan Uesato, and Pushmeet Kohli. Training verified learners with learned verifiers. *arXiv preprint arXiv:1805.10265*, 2018.

[17] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. Hotflip: white-box adversarial examples for text classification. *ACL*, 2018.

[18] Logan Engstrom, Andrew Ilyas, and Anish Athalye. Evaluating and understanding the robustness of adversarial logit pairing. *NeurIPS 2018 Workshop on Security in Machine Learning*, 2018.

[19] Yoav Freund. A more robust boosting algorithm. *arXiv preprint, arXiv:0905.2138v1*, 2009.

[20] Yoav Freund and Robert E Schapire. Experiments with a new boosting algorithm. *ICML*, 1996.

[21] Jerome Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29:1189–1232, 2001.

[22] Jerome Friedman. Stochastic gradient boosting. *Computational statistics & data analysis*, 38:367–378, 2002.

[23] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28:337–407, 2000.

[24] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *ICLR*, 2015.

[25] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *NeurIPS Workshop on Security in Machine Learning*, 2018.

[26] Chuan Guo, Jacob R Gardner, Yurong You, Andrew Gordon Wilson, and Kilian Q Weinberger. Simple black-box adversarial attacks. *ICML*, 2019.

[27] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2019. URL `http://www.gurobi.com`.

[28] Matthias Hein and Maksym Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. *NeurIPS*, 2017.

[29] Matthias Hein, Maksym Andriushchenko, and Julian Bitterwolf. Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem. *CVPR*, 2019.

[30] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. *ICML*, 2018.

[31] Harini Kannan, Alexey Kurakin, and Ian Goodfellow. Adversarial logit pairing. *arXiv preprint arXiv:1803.06373*, 2018.

[32] Alex Kantchelian, JD Tygar, and Anthony Joseph. Evasion and hardening of tree ensemble classifiers. *ICML*, 2016.

[33] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: an efficient smt solver for verifying deep neural networks. *ICCAV*, 2017.

[34] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *NeurIPS*, 2017.

[35] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[36] Bogdan Kulynych, Jamie Hayes, Nikita Samarin, and Carmela Troncoso. Evading classifiers in discrete domains with provable optimality guarantees. *NeurIPS Workshop on Security in Machine Learning*, 2018.

[37] Maksim Lapin, Matthias Hein, and Schiele Bernt. Loss functions for top-k error: analysis and insights. *CVPR*, 2016.

[38] Maksim Lapin, Matthias Hein, and Schiele Bernt. Analysis and optimization of loss functions for multiclass, top-k and multilabel classification. *PAMI*, 40:1533–1554, 2016.

[39] Hyafil Laurent and Ronald L Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 1976.

[40] Yann LeCun. The mnist database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*, 1998.

[41] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. *IEEE Symposium on Security and Privacy*, 2019.

[42] Bai Li, Changyou Chen, Wenlin Wang, and Lawrence Carin Duke. Certified adversarial robustness with addition gaussian noise. *NeurIPS*, 2019.

[43] Jiajun Lu, Hussein Sibai, Evan Fabry, and David Forsyth. No need to worry about adversarial examples in object detection in autonomous vehicles. *arXiv preprint arXiv:1707.03501*, 2017.

[44] Roman Werner Lutz, Markus Kalisch, and Peter Bühlmann. Robustified l_2 boosting. *Computational Statistics & Data Analysis*, 52:3331–3341, 2008.

[45] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *ICLR*, 2018.

[46] Seungyong Moon, Gaon An, and Hyun Oh Song. Parsimonious black-box adversarial attacks via efficient combinatorial optimization. *ICML*, 2019.

[47] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. *CVPR*, 2016.

[48] Marius Mosbach, Maksym Andriushchenko, Thomas Trost, Matthias Hein, and Dietrich Klakow. Logit pairing methods can fool gradient-based attacks. *NeurIPS 2018 Workshop on Security in Machine Learning*, 2018.

[49] Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan. Do deep generative models know what they don't know? *ICLR*, 2019.

[50] Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17:527–566, 2017.

[51] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: high confidence predictions for unrecognizable images. *CVPR*, 2015.

[52] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint, arXiv:1809.03008*, 2016.

[53] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. *IEEE EuroS&P*, 2016.

[54] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. *ICLR*, 2018.

[55] Ryan Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141, 2004.

[56] Paolo Russu, Ambra Demontis, Battista Biggio, Giorgio Fumera, and Fabio Roli. Secure kernel machines against evasion attacks. *ACM workshop on AI and security*, 2016.

[57] Hadi Salman, Greg Yang, Jerry Li, Pengchuan Zhang, Huan Zhang, Ilya Razenshteyn, and Sebastien Bubeck. Provably robust deep learning via adversarially trained smoothed classifiers. *NeurIPS*, 2019.

[58] Robert E Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 1999.

[59] Uri Shaham, Yutaro Yamada, and Sahand Negahban. Understanding adversarial training: Increasing local stability of supervised models through robust optimization. *Neurocomputing*, 2018.

[60] Jack W Smith, JE Everhart, WC Dickson, WC Knowler, and RS Johannes. Using the adap learning algorithm to forecast the onset of diabetes mellitus. *Annual Symposium on Computer Application in Medical Care*, 1988.

[61] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 2012.

[62] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *ICLR*, 2014.

[63] Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. *ICLR*, 2019.

[64] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. *ICLR*, 2019.

[65] Andrew V Uzilov, Joshua M Keegan, and David H Mathews. Detection of non-coding rnas on the basis of predicted secondary structure formation free energy change. *BMC Bioinformatics*, 7:1–30, 2006.

[66] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. *CVPR*, 2001.

[67] Paul Viola and Michael J Jones. Robust real-time face detection. *IJCV*, 57:137–154, 2004.

[68] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. *NeurIPS*, 2018.

[69] Yizhen Wang, Somesh Jha, and Kamalika Chaudhuri. Analyzing the robustness of nearest neighbors to adversarial examples. *ICML*, 2018.

[70] Manfred K. Warmuth, Karen Glocer, and Gunnar Rätsch. Boosting algorithms for maximizing the soft margin. *NeurIPS*, 2007.

[71] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Duane Boning, Inderjit S. Dhillon, and Luca Daniel. Towards fast computation of certified robustness for relu networks. *ICML*, 2018.

[72] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. *ICML*, 2018.

[73] Eric Wong, Frank Schmidt, Jan Hendrik Metzen, and J Zico Kolter. Scaling provable adversarial defenses. *NeurIPS*, 2018.

[74] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[75] Kai Y Xiao, Vincent Tjeng, Nur Muhammad Shafiullah, and Aleksander Madry. Training for faster adversarial robustness verification via inducing relu stability. *ICLR*, 2019.

[76] H. Xu, C. Caramanis, and S. Mannor. Robustness and regularization of support vector machines. *Journal of Machine Learning Research*, 10:1485–1510, 2009.

[77] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. *NeurIPS*, 2018.

A Proof of convexity of the robust objective

Lemma 1 *Let $L : \mathbb{R} \to \mathbb{R}$ be a convex, monotonically decreasing function. Then $\tilde{L} : \mathbb{R}^l \to \mathbb{R}$ defined as $\tilde{L}(u) = \max_{\tilde{x} \in C} L(c + \langle r(\tilde{x}), u \rangle)$ is convex for any $c \in \mathbb{R}$, $r : \mathbb{R}^d \to \mathbb{R}^l$, and $C \subseteq \mathbb{R}^d$.*

Proof. First we use the fact that L is monotonically decreasing:

$$\tilde{L}(u) = \max_{\tilde{x} \in C} L(c + \langle r(\tilde{x}), u \rangle) = L(c + \min_{\tilde{x} \in C} \langle r(\tilde{x}), u \rangle)$$

Now we observe that $\min_{\tilde{x} \in C} \langle r(\tilde{x}), u \rangle$ is a concave function as a pointwise minimum of a set of concave (linear) functions (see [4] regarding this property). The convexity of $\tilde{L}$ follows from the fact that it is a composition of a convex, nonincreasing function L and a concave function $c + \min_{\tilde{x} \in C} \langle r(\tilde{x}), u \rangle$. □

B Detailed algorithms

B.1 The efficient exact certification for boosted stumps

Algorithm 1: The efficient exact certification for boosted stumps

Input: ensemble of stumps $\{f_i\}_{i=1}^T$, point $x \in \mathbb{R}^d$, label $y \in \{-1, 1\}$, radius of l_∞-ball ϵ
Output: $is\_robust \in \{0, 1\}$

$G \leftarrow 0$ /* initialize the variable that will be the solution of (3) */
for $k \leftarrow 1$ **to** d **do**
  $\mathcal{F} = \{f \in \{f_i\}_{i=1}^T \mid c_s = k\}$ /* all stumps that split coord. k */
  $\delta_k^* = \texttt{CalculateMinimizer}G_k(\mathcal{F}, x, y, \epsilon)$
  $G \leftarrow G + \mathcal{F}(x_k + \delta_k^*)$
end
$is\_robust = \mathbb{1}_{G \geq 0}$

Function $\texttt{CalculateMinimizer}G_k(\mathcal{F}, x, y, \epsilon)$
  $\mathcal{F} \leftarrow$ merge the stumps in $\mathcal{F}$ with the same splitting thresholds
  $B \leftarrow \{x_k - \epsilon\},\ W \leftarrow \{0\}$
  for $s \leftarrow 1$ **to** $|\mathcal{F}|$ **do**
    /* add all thresholds and weights w_r in $[x_k - \epsilon, x_k + \epsilon]$ */
    $b_s \leftarrow \mathcal{F}_s.b,\ \ w_r^{(s)} \leftarrow \mathcal{F}_s.w_r$
    if $x_k - \epsilon < b_s \leq x_k + \epsilon$ **then**
      $B \leftarrow B \cup \{b_s\}, W \leftarrow W \cup \{w_r^{(s)}\}$
  end
  /* sorting thresholds in B leads to $O(T \log T)$ complexity */
  $\pi = \text{argsort}(B)$
  $v^* \leftarrow 0$ /* initialize the minimum cumulative difference */
  $\delta_k^* \leftarrow -\epsilon$ /* initialize the optimal perturbation for coord. k */
  for $i \leftarrow 1$ **to** $|\pi|$ **do**
    $v \leftarrow v + y\, W_{\pi_i}$
    if $v < v^*$ **then**
      $v^* \leftarrow v$, $\delta_k^* \leftarrow B_{\pi_i} - x_k$
  end
  return δ_k^*
end

B.2 Exact adversarial examples for boosted stumps

Using the result from Section 3.1, we can directly obtain provably minimal adversarial examples. By noting that the function $H(\epsilon) \coloneqq \min_{\|\delta\|_\infty \leq \epsilon} yF(x + \delta)$ is piece-wise constant with up to $T + 1$ constant

regions, it suffices to solve this minimization problem for every $\epsilon \in \{0\} \cup \{|b_t - x_{c_t}| + \nu \operatorname{sign}(b_t - x_{c_t}) \mid t = 1, \dots, T\}$ (where ν is as small as precision allows) sorted in ascending order and stop when ϵ is large enough to change the original class. In order to get the final perturbation vector δ, we have to save the indices δ_j^* that minimize $yF(x + \delta)$ for every splitting coordinate j which are used in the ensemble. The complexity of this procedure is $O(T^2 \log T)$ since in the worst case we have to solve (4) T times. For details of the procedure we refer to Algorithm 2. We provide visualizations of these exact adversarial examples in Figure 11.

Algorithm 2: Finding exact adversarial examples for boosted stumps

Input: ensemble of stumps $\{f_i\}_{i=1}^T$, point $x \in \mathbb{R}^d$, label $y \in \{-1, 1\}$
Output: exact adversarial perturbation $\delta \in \mathbb{R}^d$

$\mathcal{E} \leftarrow \{0\} \cup \{|b_t - x_{c_t}| + \nu \operatorname{sign}(b_t - x_{c_t}) \mid t = 1, \dots, T\}$
$\mathcal{E} \leftarrow \text{sort}(\mathcal{E})$
for $i \leftarrow 1$ **to** $|\mathcal{E}|$ **do**
  $\epsilon \leftarrow \mathcal{E}_i$
  $G_\epsilon \leftarrow 0$
  $\delta \leftarrow \mathbf{0}$ `/* initialize the adversarial perturbation */`
  for $k \leftarrow 1$ **to** d **do**
    $\mathcal{F} = \{f \in \{f_i\}_{i=1}^T \mid c_s = k\}$ `/* all stumps that split coord. k */`
    $\delta_k^* \leftarrow$ `CalculateMinimizer`$G_k(\mathcal{F}, x, y, \epsilon)$ `/* from Algorithm 1 */`
    $G_\epsilon \leftarrow G_\epsilon + \mathcal{F}(x_k + \delta_k^*)$
  end
  if $G_\epsilon < 0$ **then**
    break
end

B.3 Coordinate descent for the exponential loss

Boosted decision stumps: If we denote $\mathbb{1}_i$ to be equal to 1 if the first condition of (5) is true, and 0 otherwise, and also define

$$\gamma_i = \exp\big(-\sum_{k \neq j} G_k(x_i, y_i) - \sum_{s \in S_j} y_i w_l^{(s)} - h_r(x_{ij}, y_i)\mathbb{1}_i - h_l(x_{ij}, y_i)(1 - \mathbb{1}_i)\big),$$

then the total exponential loss can be written as $L(w_l, w_r) = \sum_{i=1}^n \gamma_i \exp\big(-y_i w_l - y_i w_r \mathbb{1}_i\big)$. We further denote $\mathbb{1}_{y_i = y} = \begin{cases} 1 & \text{if } y_i = y \\ 0 & \text{if } y_i \neq y \end{cases}$ and

$$\begin{aligned} \Sigma_{1,1} &= \sum_{i=1}^n \mathbb{1}_i \mathbb{1}_{y_i=1} \gamma_i \qquad & \Sigma_{1,-1} &= \sum_{i=1}^n \mathbb{1}_i \mathbb{1}_{y_i=-1} \gamma_i \\ \Sigma_{0,1} &= \sum_{i=1}^n (1 - \mathbb{1}_i) \mathbb{1}_{y_i=1} \gamma_i & \Sigma_{0,-1} &= \sum_{i=1}^n (1 - \mathbb{1}_i) \mathbb{1}_{y_i=-1} \gamma_i \end{aligned} \tag{12}$$

Then the coordinate descent update for w_l can be derived by setting $\frac{\partial L}{\partial w_l}$ to zero and solving for w_l which yields:

$$w_l = \frac{1}{2} \ln\big(\exp(-w_r)\Sigma_{1,1} + \Sigma_{0,1}\big) - \frac{1}{2} \ln\big(\exp(w_r)\Sigma_{1,-1} + \Sigma_{0,-1}\big)$$

Thus, the overall complexity for a particular coordinate j and fixed threshold b is $O(n)$ times the number of iterations of coordinate descent which is logarithmic in the desired precision (cost for bisection).

Boosted decision trees: By using the notation from (12), where now $\mathbb{1}_i \coloneqq \mathbb{1}(x_i, y_i; w_r)$, the minimizers of w_r and w_l are given by setting $\frac{\partial L}{\partial w_r}$ and $\frac{\partial L}{\partial w_l}$ to zero:

$$w_r = \frac{1}{2}\ln(\Sigma_{1,1}) - \frac{1}{2}\ln(\Sigma_{1,-1}) - w_l$$
$$w_l = \frac{1}{2}\ln\left(\exp(-w_r)\Sigma_{1,1} + \Sigma_{0,1}\right) - \frac{1}{2}\ln\left(\exp(w_r)\Sigma_{1,-1} + \Sigma_{0,-1}\right)$$

We iterate these updates of w_r and w_l until convergence. Note that coordinate descent does not create a significant overhead to the overall algorithm, since we perform only operations on scalars $\Sigma_{1,1}$, $\Sigma_{1,-1}$, $\Sigma_{0,1}$, $\Sigma_{0,-1}$ which do not have to be recomputed over the iterations of the coordinate descent.

B.4 Tree-wise certification of boosted decision trees

Algorithm 3: Tree-wise certification of boosted decision trees

Input: tree ensemble $\{f_t\}_{t=1}^T$, point $x \in \mathbb{R}^d$, label $y \in \{-1, 1\}$
Output: $is\_provably\_robust \in \{0, 1\}$

$\tilde{G} = 0$
for $t \leftarrow 1$ **to** T **do**
  $\tilde{G} = \tilde{G} +$ `ExactTreeCertification(`f_t, x, y`)`
end
$is\_provably\_robust = \mathbb{1}_{\tilde{G} \geq 0}$
Function `ExactTreeCertification(`f, x, y`)`
  `/* start from a set that contains only the root node` f `*/`
  $nodes\_to\_check = \{f\}$
  $v^* = \infty$
  while $nodes \neq \{\}$ **do**
    `/* retrieve a node and delete it from the set */`
    $node = nodes.pop()$
    `/* get the splitting coordinate of the current node */`
    $j = node.split\_coordinate$
    if $x_j \leq b + \epsilon$ **then**
      if $node.left$ *is empty* **then**
        $v^* \leftarrow \min(v^*, y \cdot node.w_l)$
      else
        $nodes\_to\_check \leftarrow nodes\_to\_check \cup \{node.left\}$
    if $x_j \geq b - \epsilon$ **then**
      if $node.right$ *is empty* **then**
        $v^* \leftarrow \min(v^*, y \cdot node.w_l + y \cdot node.w_r)$
      else
        $nodes\_to\_check \leftarrow nodes\_to\_check \cup \{node.right\}$
  end
  return v^*
end

C Monotonic descent of the upper bound on the robust loss with the shrinkage parameter

As introduced in [23], the shrinkage parameter is applied during training as follows. Let f be a new weak learner, then instead of adding it directly to the ensemble $F := F + f$, one rather adds $F := F + \alpha f$ where $\alpha \in (0, 1]$. In order to show that this scheme also always leads to monotonic descent of the upper bound on the robust loss, we apply Lemma 1 to the case where f is a decision tree with l leaves, i.e. $f(x) = u_{q(x)}$. Note that:

$$\tilde{L}(u) = \max_{\tilde{x} \in B_\infty(x,\epsilon)} L(\tilde{G}(x, y) + y u_{q(\tilde{x})}) = \max_{\tilde{x} \in C} L(c + \langle r(\tilde{x}), u \rangle),$$

where $c = \tilde{G}(x, y)$ is the contribution of the previous weak learners (see Equation (8)), $r(x) \in \{-1, 0, 1\}^l$ represents mutually exclusive boolean conditions of the tree f multiplied by the label y, i.e. $r(x)_{q(x)} = y$ and $r(x)_i = 0$ for every $i \neq q(x)$. Thus, the robust loss $\tilde{L}(u)$ is convex in the leaf weights u.

Note that $\tilde{L}(\mathbf{0})$ corresponds to the loss value when all weights of the new weak learner f are set to zero, thus it is simply the loss of the previous ensemble. Since $\tilde{L}(u)$ is convex in its leaf weights $u \in \mathbb{R}^l$, the following property holds for every $\alpha \in (0, 1]$ due to convexity of $\tilde{L}$:

$$\tilde{L}(u) < \tilde{L}(\mathbf{0}) \implies \tilde{L}(\alpha u) < \tilde{L}(\mathbf{0})$$

To see this, from the definition of convexity we have:

$$\begin{aligned} \tilde{L}\big(\alpha u + (1 - \alpha)\mathbf{0}\big) &\leq \alpha \tilde{L}(u) + (1 - \alpha)\tilde{L}(\mathbf{0}) \\ \tilde{L}(\alpha u) &\leq \alpha\big(\tilde{L}(u) - \tilde{L}(\mathbf{0})\big) + \tilde{L}(\mathbf{0}) \\ \tilde{L}(\alpha u) &< \tilde{L}(\mathbf{0}) \end{aligned}$$

Moreover, since the sum of losses over training points is also convex in u, the same reasoning applies to the sum of upper bounds on the robust losses taken over the training set. Thus, we conclude that the usage of the shrinkage parameter α still preserves the monotonic descent in the robust objective, therefore its usage is justified within our robust optimization framework.

D The cube attack

In the main part we described how to efficiently compute *upper bounds* on the robust test error. Now we would like to also have an efficient l_∞ adversarial attack on boosted trees that would allow us to perform adversarial training. Moreover, it is also interesting to visualize adversarial examples to get a better understanding how the model makes its decisions. Concretely, the goal is to find $\delta \in \mathbb{R}^d$ that approximately minimizes the following optimization problem:

$$\min_{\|\delta\|_\infty \leq \epsilon} y F(x + \delta). \tag{13}$$

We note that while there is a vast literature of black-box adversarial attacks evaluated on neural networks [6, 30, 11, 26], query-efficiency of black-box l_∞ attacks on boosted trees is less studied [11, 9]. In this paper we do not aim to fully explore this direction since our goal is primarily *provable* robustness, i.e. how to derive and optimize *upper bounds* on the robust error. Therefore, we just introduce a simple black-box attack that empirically works well for boosted trees and is efficient enough to be applied in adversarial training. We call it *the cube attack* which is based on (1+1) evolutionary algorithm [14]. The main idea of the proposed attack is that on every iteration we try to change some random subset of coordinates and accept the change only if it decreases the functional margin $yF(\hat{x})$ for the perturbed point $\hat{x}$. On every iteration of the attack, a potential change for every coordinate is sampled randomly from $\delta_i \in \{-2\epsilon,\ 0,\ 2\epsilon\}$, and after adding such $\delta \in \mathbb{R}^d$ to the perturbed point $\hat{x}_{new} := \hat{x} + \delta$ we do a projection s.t. $\|\hat{x}_{new}\|_\infty \leq \epsilon$ (and for images also $\hat{x}_{new} \in [0, 1]^d$) is satisfied. After this we keep $\hat{x}_{new}$ if $yF(\hat{x}_{new}) < yF(\hat{x})$, otherwise we keep the old value $\hat{x}$. The full procedure is specified in Algorithm 4.

Note that the obtained adversarial example is always situated at a corner of the feasible set (which is a cube or the intersection of two cubes for image data, and hence the name of the attack). A similar

idea of considering only corners of the feasible set was also used in [46] where they could design a successful adversarial attack for neural networks. The only obvious disadvantage of this attack is that it is restricted only to the corners of the l_∞-ball. However, since the considered l_∞-balls are small, it is unlikely to have a decision region which crosses only the interior of the ball, but none of its corners. The tight lower bounds on the robust test error that we show in our experiments suggest that this is indeed true in practice. Moreover, for many models the lower and upper bounds on the robust test error are *exactly equal* which suggests that with the proposed method we can avoid using expensive combinatorial MIP solvers for large-scale classification tasks while still being able to accurately estimate the robustness of the models.

Algorithm 4: The cube attack

Input: classifier F, point $x \in \mathbb{R}^d$, label $y \in \{-1, 1\}$, number of iterations N, probability p to change a coordinate (default value: $p = 0.5$)
Output: approximate minimizer $\delta \in \mathbb{R}^d$ of (13)

$\hat{x} \leftarrow x$ `/* initialize the adversarial example */`
$v^* \leftarrow yF(x)$ `/* initialize the minimum functional margin */`
for $i \leftarrow 1$ **to** N **do**
  $\delta_i \sim$ Categorical $([-2\epsilon,\ 0,\ 2\epsilon]$ with probabilities $[p/2,\ 1-p,\ p/2])\ \ \forall i \in 1, \ldots, d$
  $\hat{x}_{new} \leftarrow$ Projection of $\hat{x} + \delta$ onto $B_\infty(x, \epsilon)$ (for images also onto $[0, 1]^d$)
  $v_{new} \leftarrow yF(\hat{x}_{new})$
  `/* if the objective is improved, keep the new point` $\hat{x}_{new}$ `*/`
  if $v_{new} < v^*$ **then**
    $\hat{x} \leftarrow \hat{x}_{new}$
    $v^* \leftarrow v_{new}$
end
$\delta \leftarrow \hat{x} - x$

E Extension of the method to multi-class setting

E.1 Certification for multi-class setting

We assume that for a multi-class classifier $F : \mathbb{R}^d \to \mathbb{R}^K$, a point $x \in \mathbb{R}^d$ is classified using $y = \underset{c=1,\ldots,K}{\arg\max} F_c(x)$. Now if $y \in \{1, \ldots, K\}$ is the correct class, then x is correctly classified if and only if

$$\min_{c \neq y} [F_y(x) - F_c(x)] > 0.$$

Then the multi-class variant of the certification procedure 3 has the following form:

$$\begin{aligned} G_{mult}(x, y) &= \min_{c \neq y} \min_{\|\delta\|_p \leq \epsilon} [F_y(x + \delta) - F_c(x + \delta)] \\ &= \min_{c \neq y} \min_{\|\delta\|_p \leq \epsilon} \left[\sum_{t=1}^{T} f_{yt}(x + \delta) - \sum_{t=1}^{T} f_{ct}(x + \delta) \right] \end{aligned} \tag{14}$$

And then analogously to the binary classification case, it is not possible to change the class via a perturbation within the l_p-ball of radius ϵ if and only if $G_{mult}(x, y) > 0$.

The crucial observation now is that in the objective of (14), we have just an ensemble of $2T$ trees (T trees for each class), which we already showed how to solve exactly for stumps using Algorithm 1, and how to lower bound for trees using Algorithm 3 in order to get a robustness guarantee. Thus, robustness certification for the multi-class setting can be done directly by reusing the same routines $K - 1$ times, i.e. for every $c \in \{1, \ldots, K\} \setminus y$, and then taking the minimum over the $K - 1$ values and comparing it to zero.

E.2 Robust training for multi-class setting

Now we discuss how to properly integrate the multi-class guarantee into training via calculating an upper bound on the robust loss.

One of the first popular multi-class versions of AdaBoost is AdaBoost.MH suggested in [58] which is essentially one-vs-all classifier if labels are mutually exclusive. Although, [23] argue that the one-vs-all scheme is suboptimal, their results show that the one-vs-all approach performs similarly to the joint cross-entropy loss, see also [37, 38] for a more recent comparison. [55] compared a wide range of multi-class methods and concluded that with proper tuning of the hyperparameters of the classifiers, one-vs-all approach does not show worse results than other more involved methods. Our experiments again confirm this observation where we found that our non-robust one-vs-all models perform similarly to the models trained with XGBoost library. Thus, we describe below how we perform provably robust training for the one-vs-all scheme.

Assuming that labels for class c and training point x_i are $y_{ci} \in \{-1, 1\}$, by the *one-vs-all scheme* we mean the following optimization problem:

$$\min_{F_1,\dots,F_K} \sum_{i=1}^{n} \sum_{c=1}^{K} L(y_{ci} F_c(x_i)) = \sum_{c=1}^{K} \min_{F_c} \sum_{i=1}^{n} L(y_{ci} F_c(x_i))$$

The crucial observation is that the objective is separable over the individual classifiers $F_1, \dots, F_K$, and thus the K classifiers can be trained completely independently. A clear advantage of such a scheme is that it can be trivially parallelized. However, it is not separable anymore if we consider the *robust one-vs-all scheme* since the same adversarial perturbation δ is shared across K losses. But we still can upper bound the sum of robust losses $L(y_{ci} F_c(x_i + \delta))$ element-wise:

$$\min_{F_1,\dots,F_K} \sum_{i=1}^{n} \max_{\|\delta\|_p \leq \epsilon} \sum_{c=1}^{K} L(y_{ci} F_c(x_i + \delta)) \leq \min_{F_1,\dots,F_K} \sum_{i=1}^{n} \sum_{c=1}^{K} \max_{\|\delta\|_p \leq \epsilon} L(y_{ci} F_c(x_i + \delta))$$

and then train K one-vs-all classifiers independently. Note that from the implementation point of view, for boosted trees with the exponential loss, the only difference compared to the binary classification scheme that we described earlier is just different per-example weights γ. Thus, this scheme can be easily implemented by reusing the same procedures described earlier. This scheme already works quite well for robust boosted trees. However, we note that it is not clear how to perform *exact* robust optimization for boosted stumps for the original robust one-vs-all objective.

F Experimental details

Datasets: All datasets used in the experiments are listed in Table 4.

Table 4: Information about the datasets used in the experiments.

Dataset	# classes	# features	# train	# test	Reference
breast-cancer	2	10	546	137	[15]
diabetes	2	8	614	154	[60]
cod-rna	2	8	59535	271617	[65]
MNIST 1-5	2	784	12163	2027	[40]
MNIST 2-6	2	784	11876	1990	[40]
FMNIST shoes	2	784	12000	2000	[74]
GTS 100-rw	2	3072	4200	1380	[61]
GTS 30-70	2	3072	2940	930	[61]
MNIST	10	784	60000	10000	[40]
FMNIST	10	784	60000	10000	[74]
CIFAR-10	10	3072	50000	10000	[35]

Hyperparameters; For the breast-cancer dataset, we select the radius ϵ of the l_∞-perturbations based on the choice of [9]. However, for diabetes and cod-rna datasets we reduce them compared to [9] in a way that allows robust classifiers to still achieve a test error comparable to normal models. For image datasets (MNIST, FMNIST, GTS, CIFAR-10), we follow the established l_∞ ϵ's from the neural networks literature [72, 25].

We tune the hyperparameter w_{max} on the validation sets of several datasets and the best value came out to be close to 1, which we use for all experiments. For binary classification, we use the shrinkage

parameter of 0.2 for diabetes, cod-rna, MNIST 1-5, MNIST 2-6, and FMNIST shoes, and 0.01 for the rest of the datasets. We use at most 300 iterations for stumps, 300 iterations for trees of depth 2, 150 iterations for depth 4, and 75 iterations for trees of depth 8. For trees, we perform splits only when there are more than 10 examples at a leaf for binary classification datasets, and if more than 200 examples for multi-class datasets.

Restricting the maximum weight: In the process of fitting a decision stump (also as an intermediate step for building a tree), we have to take care of cases when all points at some side of the threshold b have the same label. This leads to w_l or w_r that attain their optimal values at $\pm\infty$ depending on the labels. In order to resolve this, in our implementation we set the maximum weight w_{max}, and we project all obtained leaf values w_l and $w_l + w_r$ onto the range $[-w_{max}, w_{max}]$. We found empiricially that constraining the maximum values of tree leafs in this way leads to a noticeable beneficial regularization effect which is similar in spirit to the usage of the shrinkage parameter introduced in [22].

G Additional experiments

G.1 Adversarial training for boosted stumps

We show the results of adversarial training for boosted stumps in in Table 5, where adversarial examples were generated using the cube attack with 10 iterations and $p = 0.5$. We observed that we could achieve non-trivial robustness (RTE) with adversarially trained models only when we used a small shrinkage parameter. Thus, we set it to 0.1 for all boosted stump models.

The results show that similarly to boosted trees, both robust training of Chen et al. [9] and our proposed methods outperform adversarial training by a large margin. This shows that either one has to find a better way to perform adversarial training for boosted stumps and trees, or that it may not be a suitable technique for classifiers which are built in a stagewise fashion.

G.2 Comparison to the robust training of Chen et al. [9]

We compare our provably robust boosted stumps and trees to Chen et al. [9] in the same setting as ours: we fit boosted stumps and boosted trees of depth 4 with 80% of the training data and use the rest as the validation set for model selection. For the models of Chen et al. [9] we use exact RTE via MIP of [32] for model selection both for stumps and trees, whereas for our models we use exact RTE for stumps, and our fast URTE for trees. For [9] we use a coarser grid for selecting the number of iterations since RTE, in particular for trees, is more expensive to evaluate. We use up to 300 iterations and shrinkage parameter of 1 for boosted stumps both for us and [9]. For boosted trees of [9] we use the number of iterations and the shrinkage parameters for every dataset separately as specified in the code of [9], and for our models as described in the previous section.

Boosted stumps: We present the results in Table 6. We can see that our robust trees lead to better RTE on 7 out of 8 datasets while having comparable test error. Moreover, our efficient way of

Table 5: The results of adversarially trained boosted stumps, where adversarial examples were generated using the cube attack. The results for other training methods are presented in Table 1. We conclude that our proposed robust stumps outperform adversarial training by a large margin.

		Adversarially trained stumps		
Dataset	l_∞ ϵ	TE	RTE	URTE
breast-cancer	0.3	0.7	15.3	15.3
diabetes	0.05	27.3	33.1	33.1
cod-rna	0.025	22.8	26.1	26.1
MNIST 1-5	0.3	3.2	8.3	9.1
MNIST 2-6	0.3	9.7	22.5	24.6
FMNIST shoes	0.1	8.3	16.3	17.0
GTS 100-rw	8/255	2.2	7.7	7.9
GTS 30-70	8/255	19.1	28.8	31.0

Table 6: Comparison of our boosted stumps to Chen et al. [9]. The model selection of the number of iterations *#iter* was done based on RTE. *Time MIP* and *Time ours* correspond to the time needed to calculate RTE of our models using a general-purpose MIP solver and our fast exact certification procedure described in Section 4.1. All numbers are obtained using full test sets.

		Stumps of Chen et al. [9]			Our robust stumps (exact robust loss)					
Dataset	l_∞ ϵ	TE	RTE	#iter	TE	RTE	#iter	Time MIP	Time ours	Speedup
breast-cancer	0.3	8.8	16.8	1	5.1	**10.9**	2	0.1s	**0.2ms**	**529x**
diabetes	0.05	23.4	**30.5**	3	27.3	31.8	1	0.1s	**0.3ms**	**393x**
cod-rna	0.025	11.6	23.2	4	11.2	**22.6**	16	6.5m	**69ms**	**5655x**
MNIST 1-5	0.3	0.9	5.2	40	0.7	**3.6**	274	37.7s	**0.14s**	**267x**
MNIST 2-6	0.3	2.8	13.9	40	3.0	**9.2**	83	14.5s	**48ms**	**302x**
FMNIST shoes	0.1	7.1	22.2	10	5.7	**10.8**	174	23.9s	**91ms**	**260x**
GTS 100-rw	8/255	2.0	11.8	40	2.0	**6.7**	109	8.1s	**0.10s**	**80x**
GTS 30-70	8/255	12.7	28.2	40	12.9	**27.6**	227	20.9s	**0.47s**	**45x**

Table 7: Comparison of our boosted trees to Chen et al. [9]. The model selection of the number of iterations *#iter* was done based on RTE for the models of [9] and URTE for our models. *Time MIP* and *Time ours* correspond to the time needed to calculate RTE of our models using a MIP solver and URTE as described in Section 4.2. All numbers are obtained using full test sets.

		Trees of Chen et al. [9]			Our robust trees (robust loss bound)						
Dataset	l_∞ ϵ	TE	RTE	#iter	TE	RTE	URTE	#iter	Time MIP	Time ours	Speedup
breast-cancer	0.3	0.7	13.1	8	0.7	**6.6**	6.6	46	5.8s	**12ms**	**502x**
diabetes	0.05	22.1	40.3	5	27.3	**35.7**	35.7	9	1.1s	**3ms**	**343x**
cod-rna	0.025	10.2	24.2	20	6.9	**21.3**	21.4	36	31.9m	**3.5s**	**550x**
MNIST 1-5	0.3	0.3	2.9	1000	0.2	**1.3**	1.4	126	3.7m	**0.14s**	**1581x**
MNIST 2-6	0.3	0.5	6.9	1000	0.7	**3.8**	4.1	88	2.6m	**0.10s**	**1500x**
FMNIST shoes	0.1	3.1	13.2	20	3.6	**8.0**	8.1	128	3.6m	**0.14s**	**1522x**
GTS 100-rw	8/255	1.5	9.7	20	2.6	**4.7**	4.7	105	1.4m	**57ms**	**1417x**
GTS 30-70	8/255	11.5	28.8	20	13.8	**20.9**	21.4	148	2.4m	**0.10s**	**1463x**
MNIST	0.3	2.0	31.2	200	2.7	**12.5**	15.8	37	5.5 days	**4.6s**	**135893x**
FMNIST	0.1	14.4	65.1	200	14.2	**23.2**	25.9	52	3.3 days	**4.2s**	**82209x**

calculating the RTE described in Section 3.1 is orders of magnitude faster than using an off-the-shelf MIP-solver [27]. We note that the most robust models of [9] are usually obtained at the first 40 iterations, while our models need more iterations to obtain the minimum validation RTE. We attribute this to the differences in robust training and also to the fact that we use a different loss function and constrain w_{max}. We observe that the latter usually increases the number of iterations needed until convergence.

Boosted trees: First, we note that in order to make the MIP formulation of [32] more scalable for tree ensembles, we change it to the feasibility problem regarding whether there exists an l_∞-perturbation that is able to change the class instead of searching for the *minimal* adversarial perturbation wrt the l_∞-norm. This brings us in average two orders of magnitude speed-up for calculating RTE on the considered datasets. However, even with this speed-up, it still takes up to 5.5 days to calculate RTE for the largest models that we evaluated.

We present the comparison for boosted trees in Table 7. The main observation is that we outperform [9] on *all* considered datasets in terms of the RTE, often by a large margin. Our better RTE comes at the price of slightly worse test error on several datasets which we attribute to the empirically observed trade-off between accuracy and robustness: methods achieving better robustness tend to have worse test error. We note that our URTE are very close to RTE, and the time needed to calculate URTE is orders of magnitude faster than RTE calculated with MIP.

In Table 7 we also provide a comparison for boosted trees on multi-class datasets (MNIST and FMNIST). We trained our models using the one-vs-all approach and set the depth of individual trees to be up to 30. For [9] we take the models provided by the authors that have depth 8. We can see that our robust trees outperform their method by a large margin: 12.5% instead of 31.2% RTE on MNIST. On FMNIST, the gap is even larger: 23.2% versus 65.1% RTE while our test error is even slightly

better. We note that partially the reason for such a large gap might be in the fact that the boosted trees of [9] may also benefit from a larger depth. However, our comparison for binary classification datasets suggests that even when the settings are the same for both methods, our robust training consistently leads to more robust models than [9].

G.3 Robust boosted trees of different depth

The results for boosted trees are given in Table 8 for trees of different depth. We show lower bounds on robust test error (LRTE) obtained via the cube attack to show that it leads to tight LRTE which are close to the exact RTE values. This justifies its usage in adversarial training. For LRTE we used the attack with 20 iterations and $p = 0.5$. We run the attack every iteration of training, and initialize every next perturbation δ with the perturbation obtained at the previous iteration. We perform l_∞ adversarial training similarly to [32], i.e. every iteration we train on clean training points and adversarial examples (equal proportion), which are generated via the cube attack using 10 iterations and $p = 0.5$.

We observe that robust training for boosted trees is very efficient in improving robustness of the models for all depth values. In particular, our robust models outperform adversarially trained models, often with a large margin. For example, on MNIST 1-5, RTE of the adversarially trained model of depth 8 is 10.5%, while RTE of our robust model of the same depth is 1.2%. We observe that for our robust trees, URTE is very close to LRTE or even the same in some cases which can allow us to assess exact RTE even without using any combinatorial solvers. Finally, we note that our trees of depth 4 outperform our trees of depth 2 on all datasets in terms of RTE. However, our models of depth 8 show a better RTE than depth 4 only on several datasets including MNIST 1-5 and MNIST 2-6. For MNIST and FMNIST we observed improvements in RTE by increasing the depth up to 30. This suggests that in order to achieve the optimal RTE, one has to carefully select an appropriate depth of the trees which depends on a particular dataset.

G.4 Robustness and accuracy

There is a lot of empirical evidence that robust training methods for neural networks exhibit a trade-off between robustness and accuracy [73, 25, 64]. Now we investigate whether the same trade-off also exists for our robust boosted trees. For this we take three datasets (diabetes, cod-rna, and FMNIST shoes) and plot the dependency of the test error on l_∞ ϵ used for our robust training. The results are presented in Figure 4 for trees of depth 4 and 8. We can confirm that the trade-off can also be observed for boosted trees: we consistently lose accuracy once we increase ϵ. The only *slight* gain in accuracy that we observe is on FMNIST shoes dataset.

G.5 Feature importance

It is important to note that boosted trees that split directly on pixel values are not the most suitable models for computer vision tasks. Even though on some datasets like GTS 100-rw, they are able to achieve less than 1% test error, they lack important invariances such as invariance to translations, different view points, etc. What we would like to emphasize in this section is the advantage of boosted trees in terms of *transparent decision making*. In particular, we can clearly see which pixels are directly used for the decisions. One of the ways to assign feature importance to boosted decision trees with coordinate-aligned splits is to count the number of times a particular feature was used in some splits. Such visualization are shown in Figures 5, 6, 7. First of all, we can note that for all datasets our robust training changes the frequencies of features that are used. For example, on the breast cancer dataset, the robust model tends to use features like texture, concave points, area, radius, and compactness much less often compared to the normal and adversarially trained models. On MNIST 1-5 and MNIST 2-6 we see that the robust model relies more often at the pixels which are closer to the border. On GTS 100-rw and GTS 30-70 all the models rely mainly just on a few discriminative pixels (see Figure 13 for examples of the images). It is particularly interesting that on GTS 100-rw the models can achieve almost perfect classification error while ignoring almost the whole image. This shows that even a good performance on some test set does not yet mean that the model has truly learned important features – just shifting the GTS images by several pixel would completely ruin the performance of the presented boosted tree models. Thus we again emphasize the importance of interpretability for detecting such failure modes.

Table 8: Evaluation of robustness for boosted trees of different depth. We show, in percentage, test error (TE), lower bound on robust test error (LRTE) via the cube attack, robust test error (RTE) via MIP of [32], upper bound on robust test error (URTE), and the number of iterations selected using the validation set (#iter). Our robust boosted trees significantly improve RTE, more than adversarially trained boosted trees. We also observe that URTE is close to RTE for many models.

		Normal trees (standard training)					Adversarially trained trees (with cube attack)					Our robust trees (robust loss bound)				
Dataset	l_∞ ϵ	TE	LRTE	RTE	URTE	#iter	TE	LRTE	RTE	URTE	#iter	TE	LRTE	RTE	URTE	#iter
depth=2																
breast-cancer	0.3	1.5	81.0	81.0	82.5	47	**0.7**	29.2	29.2	29.2	3	2.2	**10.2**	**10.2**	**10.2**	12
diabetes	0.05	**22.7**	43.5	44.8	45.5	20	25.3	38.3	38.3	38.3	3	28.6	**36.4**	**36.4**	**36.4**	20
cod-rna	0.025	**3.9**	35.6	37.0	39.2	298	11.5	22.9	22.9	22.9	2	7.2	**21.6**	**21.6**	**21.6**	229
MNIST 1-5	0.3	**0.1**	57.5	88.5	99.0	192	1.9	8.6	8.8	9.1	7	0.5	**1.8**	**1.8**	**1.8**	140
MNIST 2-6	0.3	**0.7**	95.5	100	100	276	4.7	17.5	17.5	17.5	8	1.2	**4.8**	**4.8**	**5.0**	291
FMNIST shoes	0.1	**1.6**	95.6	100	100	268	6.6	13.3	13.5	13.8	15	4.4	**8.5**	**8.6**	**8.6**	137
GTS 100-rw	8/255	5.1	13.4	13.4	13.4	234	12.6	18.7	19.0	19.0	69	**3.8**	**7.8**	**7.8**	**7.8**	299
GTS 30-70	8/255	17.0	29.4	29.4	29.7	300	22.3	27.5	28.8	28.8	153	**15.9**	**23.4**	**23.4**	**23.6**	292
depth=4																
breast-cancer	0.3	0.7	81.0	81.0	81.8	78	**0.0**	19.7	27.0	27.0	3	0.7	**6.6**	**6.6**	**6.6**	46
diabetes	**0.05**	**22.7**	51.3	55.2	61.7	18	26.6	45.5	46.8	46.8	1	27.3	**35.7**	**35.7**	**35.7**	9
cod-rna	0.025	**3.4**	37.6	41.6	47.1	150	10.9	24.6	24.8	24.8	2	6.9	**21.3**	**21.3**	**21.4**	36
MNIST 1-5	0.3	**0.1**	59.1	90.7	96.0	72	1.3	7.1	9.0	9.5	5	0.2	**1.3**	**1.3**	**1.4**	126
MNIST 2-6	0.3	**0.4**	89.6	89.6	100	79	2.3	15.1	15.1	15.9	6	0.7	**3.8**	**3.8**	**4.1**	88
FMNIST shoes	0.1	**1.7**	84.0	99.8	99.9	117	5.5	13.2	14.1	14.2	12	3.6	**7.7**	**8.0**	**8.1**	128
GTS 100-rw	8/255	**0.9**	5.8	6.0	6.1	148	1.0	5.7	8.4	8.4	40	2.6	**4.7**	**4.7**	**4.7**	105
GTS 30-70	8/255	14.2	31.1	31.4	32.6	148	16.2	24.7	26.7	26.8	26	**13.8**	**20.9**	**20.9**	**21.4**	148
depth=8																
breast-cancer	0.3	**0.7**	83.9	84.7	84.7	54	**0.7**	13.1	19.7	19.7	3	**0.7**	**8.8**	**8.8**	**8.8**	1
diabetes	0.05	**22.1**	68.8	83.1	91.6	27	29.9	73.4	77.9	77.9	1	27.3	**35.7**	**35.7**	**35.7**	2
cod-rna	0.025	**3.2**	38.9	49.0	61.3	72	5.6	28.9	30.8	31.8	2	6.6	**21.0**	**21.1**	**21.1**	5
MNIST 1-5	0.3	0.4	86.6	92.6	94.5	28	1.0	7.2	10.5	11.4	5	**0.2**	**1.0**	**1.2**	**1.4**	60
MNIST 2-6	0.3	**0.4**	78.1	95.1	99.9	61	0.8	9.3	11.7	12.1	7	**0.4**	**2.7**	**3.0**	**3.3**	72
FMNIST shoes	0.1	**1.8**	80.2	99.9	100	64	4.5	14.5	16.5	16.6	7	3.3	**7.4**	**8.3**	**8.3**	12
GTS 100-rw	8/255	8.7	19.6	19.7	20.8	38	**0.9**	**6.1**	13.3	13.5	32	6.0	10.5	**10.6**	**11.3**	25
GTS 30-70	8/255	15.4	39.6	40.0	40.9	39	14.3	23.2	25.5	25.8	21	**11.9**	**21.0**	**21.1**	**22.0**	63

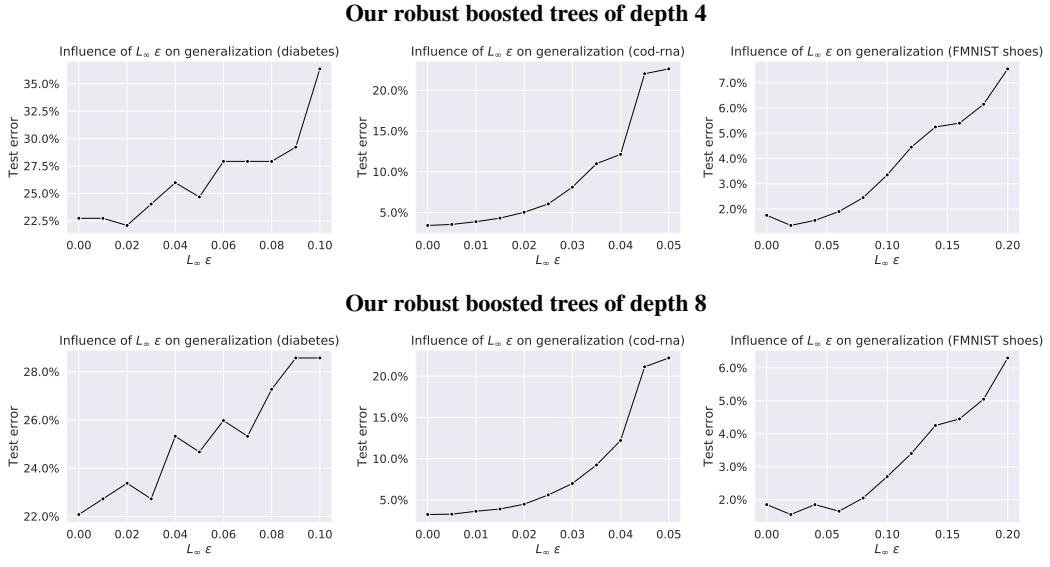

Figure 4: Robustness vs test error trade-off of our robust boosted trees. We can observe that robustness often comes with a loss in test error depending on the particular value of ϵ. However, for FMNIST shoes, there exists a range of ϵ when robust training helps to slightly improve test error.

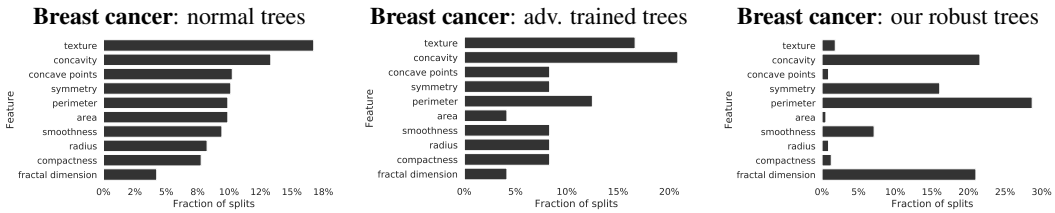

Figure 5: Feature importance of different boosted tree models on breast-cancer dataset based on the number of splits made at a particular pixel.

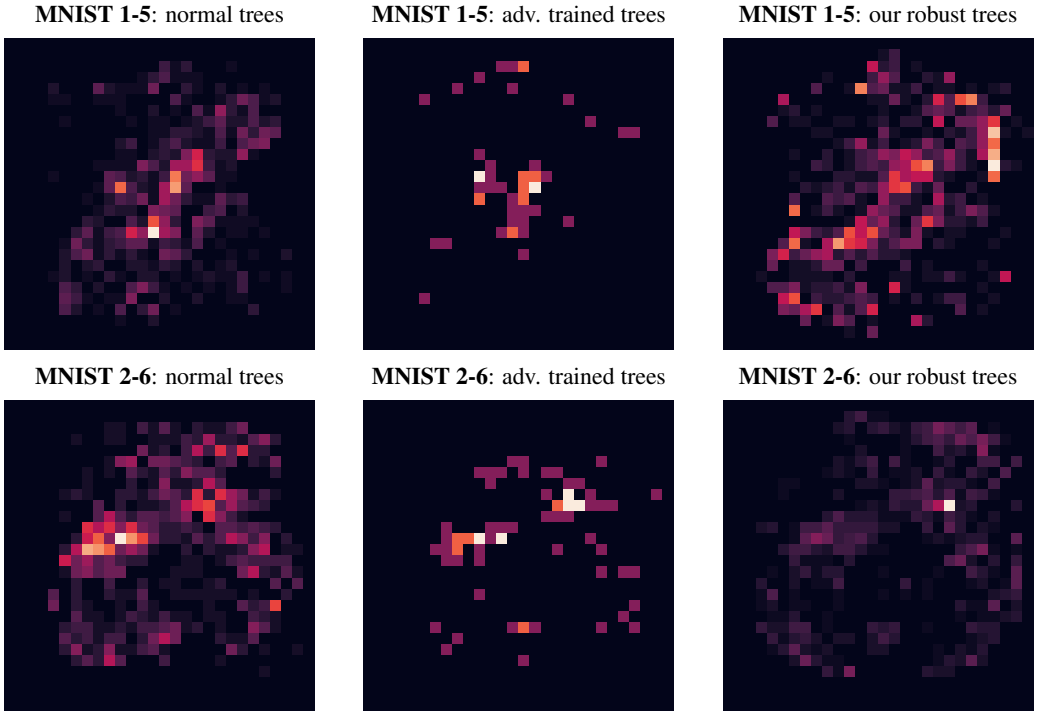

Figure 6: Feature importance of different boosted tree models on MNIST 1-5 and MNIST 2-6 based on the number of splits made at a particular pixel.

G.6 Distribution of splitting thresholds

In Figures 8, 9, 10, we plot the distibutions of the splitting thresholds b for the three boosted tree models of depth 4 on breast-cancer, MNIST 1-5, MNIST 2-6, GTS 100-rw, and GTS 30-70 datasets reported in Table 2. We can observe that our robust models on breast-cancer tend to select splits away from 0 and 1. On MNIST 1-5 and MNIST 2-6 the distributions for the normal and robust models are completely different – almost all splits for the normal model are very close to 0 and 1, while the splits for the robust model are mostly in the range between 0.3 and 0.7. This is reasonable given that more than 80% pixels of MNIST are either 0 or 1, and the considered l_∞-perturbations are within $\epsilon = 0.3$. And since the normal model splits arbitrarily close to 0 or 1, this suggests that its decisions might be easily flipped if the adversary is allowed to change them within ϵ. We also note that adversarially trained models have a distribution of the splitting thresholds that resembles the distribution for our models, however there are still quite many non-robust splits around 0 and 1. This again emphasizes the importance of solving the robust optimization problem properly. On GTS 100-rw and GTS 30-70 we can see that the distribution of thresholds for the robust model differs from the normal and adversarially trained models. It is interesting to note that there are no splits too close to one.

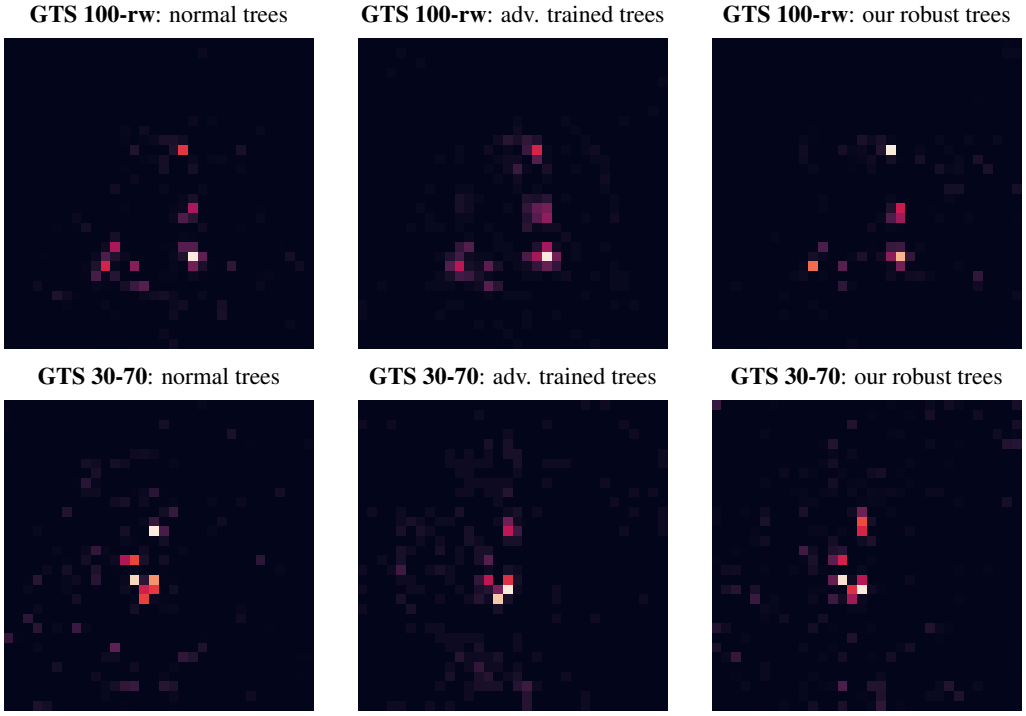

Figure 7: Feature importance of different boosted tree models on GTS 100-rw and GTS 30-70 based on the number of splits made at a particular pixel.

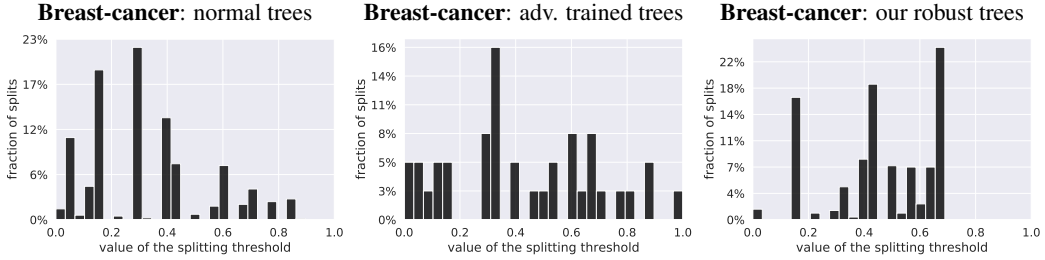

Figure 8: The distribution of the splitting thresholds for boosted tree models trained on breast-cancer dataset. We can observe that the choice of splitting thresholds is different for the robust model, in particular it does not have splits larger than at 1 - ϵ ($\epsilon = 0.3$).

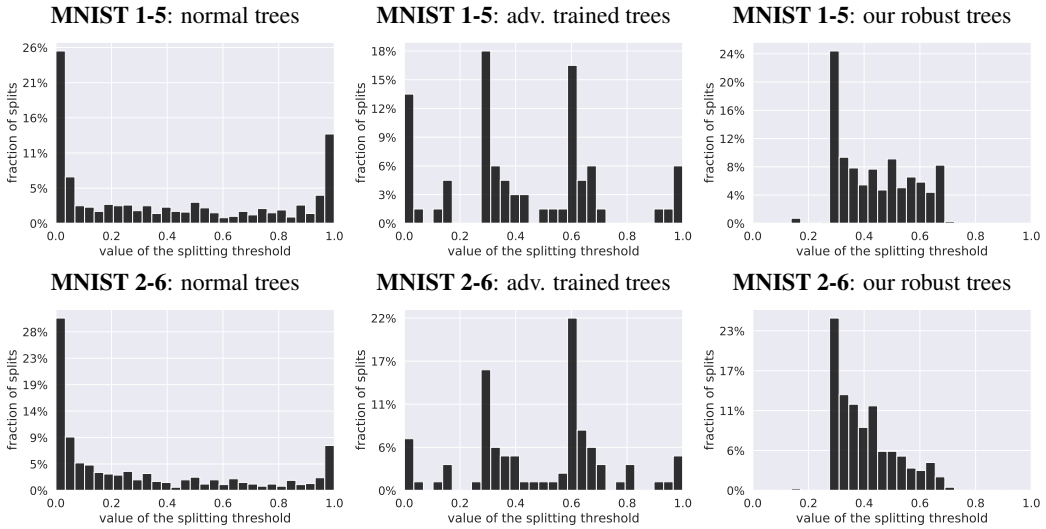

Figure 9: The distribution of the splitting thresholds for boosted tree models trained on MNIST 1-5 and MNIST 2-6. We can observe that the robust model almost always select splits in the range between 0.3 and 0.7, which is reasonable according to l_∞-perturbations within $\epsilon = 0.3$. At the same time, the normal model splits arbitrarily close to 0 or 1, which suggests that its decisions might be easily flipped by the adversary.

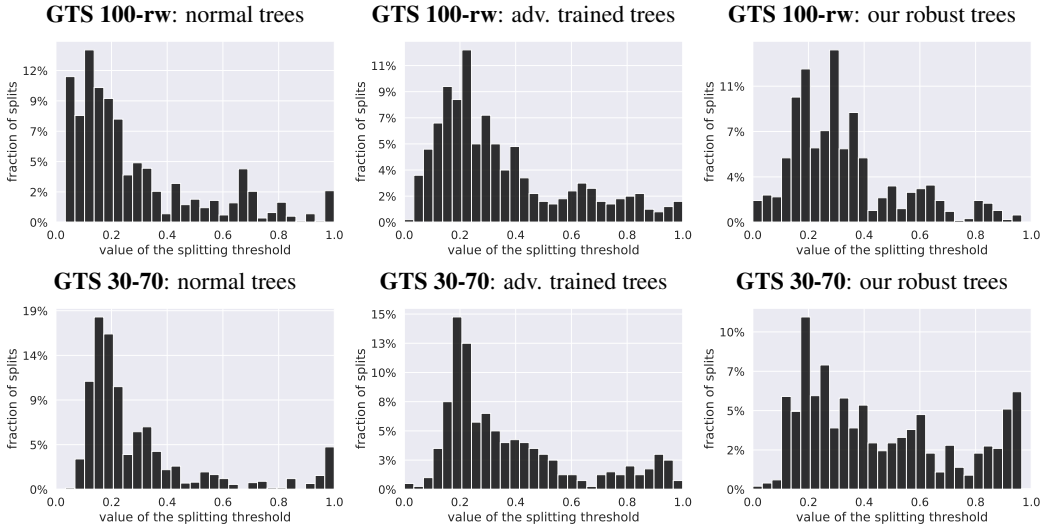

Figure 10: The distribution of the splitting thresholds for boosted tree models trained on GTS 100-rw and GTS 30-70. We can observe that the robust model often selects splits in the range between 8/255 (≈ 0.031) and 1 - 8/255 (≈ 0.969), which is reasonable according to l_∞-perturbations within $\epsilon = 8/255$.

G.7 Adversarial examples for boosted stumps and trees

Exact adversarial examples for boosted stumps: In Section 3.1, we described how we can efficiently obtain provably minimal (exact) adversarial examples for boosted stumps. We show them for MNIST 1-5 and MNIST 2-6 datasets in Figure 11. We show the size of l_∞-perturbation needed to flip the class in the title of each image. First, we can observe that l_∞-perturbations are sparse which is due to the fact that we modify only the pixels that influence particular decision stumps that contribute to minimization of (4). The main observation is that the perturbations for normal models are extremely small, while for robust models they are much larger in terms of the l_∞-norm. In particular, they have usually $\|\delta\|_\infty$ slightly larger than 0.3 which makes sense since the ϵ that we used during training was equal to 0.3. Moreover, for robust models, the perturbations are situated at the locations where one can expect pixels of the opposite classes.

Adversarial examples for boosted trees: Adversarial examples for different boosted tree models are obtained via the binary search applied on top of the cube attack. We show the resulting images in Figure 12 for MNIST 1-5 and MNIST 2-6, and in Figure 13 for GTS 100-rw and GTS 30-70. We note that qualitatively the adversarial examples for boosted trees are very similar to the exact adversarial examples for boosted stumps. Except that for a few images the perturbation is larger in l_∞-norm and affects more pixels. This might be an artifact of how the cube attack works, although for visualization purposes we remove the perturbations from the features that do not affect any splits. For GTS 100-rw and GTS 30-70, we see that the changes that flip the class are often quite small even for our robust models which is due to the fact that we used a small ϵ during training $(8/255)$ which is much lower than the ϵ for MNIST 1-5 or MNIST 2-6. We can see noticeable changes mostly for the images that have a natural contrast level. For low-contrast images the changes are harder to spot, but they are still present at the locations shown on the heatmaps from Figure 7.

Overall, we can conclude that for boosted stumps and trees the presented adversarial examples do not show perceptual interpolations between classes like robust neural networks [64], but this we cannot expect from such simple classifiers. What is more important in the context of stumps and trees is rather the idea of instance-based explanations that can help to get more insights into how the model makes its decisions.

Figure 11: Exact adversarial examples for boosted stumps trained on MNIST 1-5 and MNIST 2-6 datasets. We show the size of l_∞-perturbation needed to flip the class in the title of each image. We can observe that perturbations for normal models are extremely small or even imperceptible, while for robust models they are much larger in l_∞-norm and situated at the locations where one can expect pixels of the opposite classes.

Figure 12: Adversarial examples for boosted trees trained on MNIST 1-5 and MNIST 2-6 datasets. We show the size of l_∞-perturbation needed to flip the class in the title of each image. We can observe that perturbations for normal models are extremely small or even imperceptible, while for robust models they are much larger in l_∞-norm and situated at the locations where one can expect pixels of the opposite classes.

Figure 13: Adversarial examples for boosted trees trained on GTS 30-70 and GTS 100-rw datasets. We show the size of l_∞-perturbation needed to flip the class in the title of each image. We see that the changes are often quite small even for our robust models which is due to the fact that we used a small ϵ during training $(8/255)$ which is much lower than the ϵ for MNIST 1-5 or MNIST 2-6.