CHAPTER 16

Adversarial robustness of beyond neural network models

Adversarial robustness of neural network models have been extensively studied, and up to this chapter we were mainly discussing attack, defense and verification for neural networks. Many other continuous machine learning models, such as support vector machines (SVM) or logistic regression models, can be viewed as a special kind of neural networks, so that most of the techniques developed for neural networks can be immediately applied. However, there are many other important machine learning models that involve some "discrete" components, and the studies on neural networks cannot be directly extended to those models. For example, K-nearest neighbor (KNN) is a very important fundamental machine learning model, where prediction is made based on one or few closest samples in the training dataset. The decision boundary of KNN can be visualized as in Fig. 16.1, which is nonsmooth and nondifferentiable. Also, a decision tree or decision forest (an ensemble of multiple decision trees) makes discrete decisions based on some cut-offs of feature values, so its decision function is discrete rather than continuous. Due to the discrete properties of those models, we need different ways for attack, verification, and defense.

In this chapter, we discuss how to evaluate and improve the robustness of two important discrete nonneural network models: K-nearest-neighbor (KNN) classifiers and decision tree ensembles, including gradient boosting decision tree (GBDT) and random forest (RF) models. In general, we will show that measuring the robustness of these models can be formulated as optimization problems, similar to neural networks, despite the fact that those optimization problems involve discrete variables. We will then show that attacking those models is equivalent to finding a primal feasible solution to the optimization problem, whereas robustness verification is equivalent to finding a dual solution to give a lower bound of those optimization problems. We will also discuss how to enhance the robustness of those discrete models.

Before going to detail, we would like to emphasize that there are several good reasons to study the robustness of those discrete machine learning models, as listed below:

Adversarial Robustness for Machine Learning
https://doi.org/10.1016/B978-0-12-824020-5.00027-2

1. Although neural networks are good at extracting informative features from images/audio/text, there are many other applications where neural networks may not outperform other classical machine learning models. In particular, the gradient boosting decision tree (GBDT) model is more widely used than neural networks in many data mining tasks. Many Kaggle data mining competitions are won by GBDT models, and GBDT is still widely used in ranking systems in industry. Hence it is important to understand how to measure and improve the robustness of these discrete models.
2. Discrete structures may be more robust than neural networks since the discrete operations are less sensitive to small perturbations. Several practical defense approaches have combined nearest-neighbor classifier with neural networks (Papernot and McDaniel, 2018; Dubey et al., 2019; Sitawarin and Wagner, 2019). Studying the robustness of discrete models is thus essential for understanding whether they can provide improved robustness.

16.1 Evaluating the robustness of K-nearest-neighbor models

Nearest-neighbor (NN) classifiers predict the label of a testing instance by one or K closest samples in the database. Assume that there are C labels in total. We use $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ to denote the database where each $\boldsymbol{x}_i$ is a d-dimensional vector, and $y_i \in \{1, \ldots, C\}$ is the corresponding label. A K-NN classifier $f : \mathbb{R}^d \to \{1, \ldots, C\}$ maps a test instance to a predicted label. Given a test instance $\boldsymbol{z} \in \mathbb{R}^d$, the classifier first identifies the K-nearest neighbors $\{\boldsymbol{x}_{\pi(1)}, \ldots, \boldsymbol{x}_{\pi(K)}\}$ based on the Euclidean distance $\|\boldsymbol{x}_i - \boldsymbol{z}\|$ and then predicts the final label by majority voting among $\{y_{\pi(1)}, \ldots, y_{\pi(K)}\}$.

Given a test sample $\boldsymbol{z}$, assume that it is correctly classified as class-1 by the NN model. An adversarial perturbation is defined as $\boldsymbol{\delta} \in \mathbb{R}^d$ such that $f(\boldsymbol{z} + \boldsymbol{\delta}) \neq 1$. Robustness evaluation aims to find the minimum-norm adversarial perturbation

$$\boldsymbol{\delta}^* = \arg\min_{\boldsymbol{\delta}} \|\boldsymbol{\delta}\| \quad \text{s.t.} \quad f(\boldsymbol{z} + \boldsymbol{\delta}) \neq 1, \tag{16.1}$$

and we define $\epsilon^* = \|\boldsymbol{\delta}^*\|$ as the norm of this perturbation. In many cases it is challenging to obtain the optimal solution of (16.1), so we will discuss algorithms for attack and verification. Attack algorithms aim to find a feasible (but may be subpotimal) solution $\boldsymbol{\delta}$ for (16.1), which serves as an

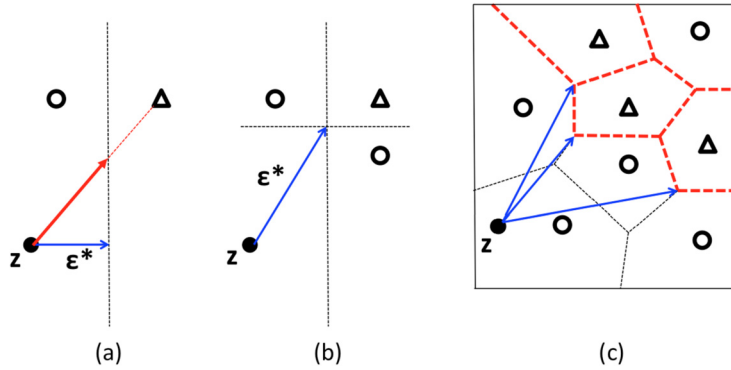

Figure 16.1 Illustration of the minimum adversarial perturbation for 1-NN model. The goal is to perturb $\boldsymbol{z}$ to be classified as triangle class instead of the circle class; (a) shows the optimal perturbation (ϵ^*) with two training points (circle and triangle); (b) shows the optimal perturbation when there are three training points; (c) shows that the optimal perturbation can be computed by evaluating the distance from $\boldsymbol{z}$ to each Voronoi cell of triangle instances.

adversarial example. A verification algorithm aims to find a lower bound r on the minimal perturbation such that

$$f(\boldsymbol{z}+\boldsymbol{\delta}) = 1 \quad \forall \|\boldsymbol{\delta}\| \leq r.$$

Clearly, by definition the maximum lower bound r^* will match the minimum perturbation norm ϵ^* if both attack and verification are optimal. When $r < r^*$, we say the verification algorithm is sound but incomplete (more discussions about complete versus incomplete verification can be found in Chapter 7.

For simplicity, we will mainly focus on the ℓ_2 norm with 1-NN model (only looking at the closest sample in the database) and then generalize to $K > 1$ later.

First, let's consider some simple examples to gain intuition what would be the optimal adversarial example for the 1-NN model. In Fig. 16.1(a), we have two points (a circle and a triangle) in the database. For the testing sample $\boldsymbol{z}$, it is predicted as a circle, and we would like to know the minimum perturbation to change the prediction (nearest neighbor becomes a triangle). A naive way is to find the perturbation on the direction pointing to the triangle, but in fact we cannot find the minimum perturbation on that direction. When there are only one circle and one triangle, the minimum perturbation will be pointing from $\boldsymbol{z}$ to the bisection hyperplane between two training samples (blue (dark gray in print version) ϵ^*

in Fig. 16.1(a)). However, when there are more than two samples, it becomes insufficient to check the projection to each bisection hyperplane. For example, in Fig. 16.1(b), there are two circles and one triangle, and the optimal perturbation (ϵ^* in blue (dark gray in print version)) is pointing to the intersection of two bisection hyperplanes (two dash lines). If we check the perturbation from $\boldsymbol{z}$ to all the intersections of bisection hyperplanes, then we will be able to find the minimum adversarial perturbation. However, as the number of intersections of bisections grows exponentially to the number of training samples, it becomes computationally intractable to conduct an exhausted search.

In fact, the decision boundary of a 1-NN model can be captured by the Voronoi diagram (see Fig. 16.1(c)). In the Voronoi diagram, each training instance $\boldsymbol{x}_i$ forms a cell, and the decision boundary of the cell is captured by the convex boundary formed by bisections between $\boldsymbol{x}_i$ and its neighbors. We can thus obtain the minimum adversarial perturbation by computing the distances from $\boldsymbol{z}$ to all the cells with $y_i \neq 1$. However, to compute the distance, we need to check all the faces (captured by one bisection hyperplane) and angles (intersections of more than one bisection hyperplanes) of the cell.

For the two-dimensional space ($d = 2$), it has been shown by Aurenhammer and Klein (1999) that each cell can only have finite faces and angles and there exists a polynomial-time algorithm for computing a Voronoi diagram. In general, for d-dimensional problems with n points, Voronoi diagram computation requires $O(n \log n + n^{\frac{d}{2}})$ time, which works for low-dimensional problems. However, the time complexity grows exponentially with dimension d, so in general, it is hard to use this algorithm unless d is very small.

A primal-dual quadratic programming formulation

Computing minimum adversarial perturbations for ReLU networks is NP-hard (Katz et al., 2017). Also, as discussed in the previous section, we can connect it to the Voronoi diagram computation, but the solver will require exponential time in dimensionality. So is it NP-hard to compute the minimum adversarial perturbation for 1-NN? Surprisingly, the answer is no, and we will describe an efficient algorithm for doing that.

For a given instance $\boldsymbol{z}$, if we want to perturb it so that $\boldsymbol{z} + \boldsymbol{\delta}$ is closer to $\boldsymbol{x}_j$ with $y_j \neq 1$ than to all class-1 instances, then the problem of finding the

minimum perturbation can be formulated as

$$\epsilon^{(j)} = \min_{\boldsymbol{\delta}} \frac{1}{2}\boldsymbol{\delta}^T\boldsymbol{\delta} \;\text{ s.t. } \|\boldsymbol{z}+\boldsymbol{\delta}-\boldsymbol{x}_j\|^2 \le \|\boldsymbol{z}+\boldsymbol{\delta}-\boldsymbol{x}_i\|^2 \;\; \forall i, y_i = 1. \tag{16.2}$$

Note that this formulation was proposed by (Wang et al., 2019c; Yang et al., 2020f). (Wang et al., 2019c) further shows that (16.2) can be solved in polynomial time as follows.

Each constraint in the formulation can be rewritten as

$$\boldsymbol{\delta}^T(\boldsymbol{x}_j - \boldsymbol{x}_i) + \frac{\|\boldsymbol{z}-\boldsymbol{x}_i\|^2 - \|\boldsymbol{z}-\boldsymbol{x}_j\|^2}{2} \ge 0.$$

Therefore (16.2) becomes

$$\epsilon^{(j)} = \min_{\boldsymbol{\delta}: A\boldsymbol{\delta}+\boldsymbol{b}\ge 0} \{\frac{1}{2}\boldsymbol{\delta}^T\boldsymbol{\delta}\} := P^{(j)}(\boldsymbol{\delta}), \tag{16.3}$$

where $A \in \mathbb{R}^{n\times d}$ and $\boldsymbol{b} \in \mathbb{R}^n$ with $y_i = 1$, $\boldsymbol{a}_i = (\boldsymbol{x}_j - \boldsymbol{x}_i)$, and $b_i = \frac{\|\boldsymbol{z}-\boldsymbol{x}_i\|^2 - \|\boldsymbol{z}-\boldsymbol{x}_j\|^2}{2}$ for each row i ($\mathbf{0}$ otherwise). By solving the quadratic programming (QP) problem (16.3) for each $\{j : y_j \neq 1\}$, the final minimum adversarial perturbation norm is $\epsilon^* = \min_{j:y_j\neq 1}\sqrt{2\epsilon^{(j)}}$. It has been shown that convex quadratic programming can be solved in polynomial time (Kozlov et al., 1980), so this formulation leads to a *polynomial-time algorithm* for finding ϵ^*. However, naively solving this will still be too expensive as the number of constraint grows quadratically. We will thus introduce the dual problems below, which not only give an efficient way for computing robustness of 1-NN model, but also provide ways to compute a certifiable robustness bound for 1-NN robustness verification.

Dual quadratic programming problems

We also introduce the dual form of each QP as it is more efficient to solve in practice and will lead to a family of verification algorithms for adversarial robustness. The dual problem of (16.3) can be written as

$$\max_{\boldsymbol{\lambda}\ge 0} \left\{-\frac{1}{2}\boldsymbol{\lambda}^T AA^T\boldsymbol{\lambda} - \boldsymbol{\lambda}^T\boldsymbol{b}\right\} := D^{(j)}(\boldsymbol{\lambda}), \tag{16.4}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^{n^+}$ are the corresponding dual variables. This follows the standard derivation of dual QP problems (e.g., almost identical to the primal and dual problems of support vector machines). The primal-dual relationship connects primal and dual variables by $\boldsymbol{\delta} = A^T\boldsymbol{\lambda}$. Based on weak duality,

we have $D^{(j)}(\boldsymbol{\lambda}) \leq P^{(j)}(\boldsymbol{\delta})$ for any dual feasible solution $\boldsymbol{\lambda}$ and primal feasible solution $\boldsymbol{\delta}$. Furthermore, based on Slater's condition, we can show that (16.3) satisfies strong duality, so $D^{(j)}(\boldsymbol{\lambda}^*) = P^{(j)}(\boldsymbol{\delta}^*)$, where $\boldsymbol{\lambda}^*$ and $\boldsymbol{\delta}^*$ are optimal solutions for the primal and dual problems, respectively, if $\boldsymbol{x}_j^- \neq \boldsymbol{x}_i^+$ for all $i \in [n^+]$.[1] Based on strong duality, we have

$$\begin{aligned}\frac{1}{2}(\epsilon^*)^2 &= \min_{j\in[n^-]} \{P^{(j)}(\boldsymbol{\delta}^*)\} \\ &= \min_{j\in[n^-]} \{\max_{\boldsymbol{\lambda}\geq 0} D^{(j)}(\boldsymbol{\lambda})\} \\ &\geq \min_{j\in[n^-]} \{D^{(j)}(\boldsymbol{\lambda}^{(j)})\} \text{ with feasible } \boldsymbol{\lambda}^{(j)}, \end{aligned} \tag{16.5}$$

so a set of feasible solutions $\{\boldsymbol{\lambda}^{(j)}\}_{j\in[n^-]}$ leads to a lower bound of the minimum adversarial perturbation. In summary, we conclude the primal-dual relationship between 1-NN attack and verification:

- A primal feasible solution of $P^{(j)}$ is a successful attack and gives us an upper bound of ϵ^*. Therefore we can solve a subset of QPs and select the minimum. Usually, $\boldsymbol{x}_j^-$ closer to $\boldsymbol{z}$ will lead to a smaller adversarial perturbation, so in practice we can sort $\boldsymbol{x}_j^-$ by the distance to $\boldsymbol{z}$, solve the subproblems one by one, and stop at any time. It will give a valid adversarial perturbation. After solving all the subproblems, the result will reach the minimum, i.e., the minimum adversarial perturbation norm ϵ^*.
- A set of dual feasible solutions $\{\boldsymbol{\lambda}^{(j)}\}_{j\in[n^-]}$ will lead to a lower bound of ϵ^* according to (16.5). Thus any heuristic method for setting up a set of dual feasible solutions will give us a lower bound, which can be used for robustness verification. If all the dual problems are solved exactly, then we will derive the tightest (maximum) lower bound, which is also ϵ^*.

Robustness verification for 1-NN models

Here we give an example of how to quickly set up dual variables to give a nontrivial lower bound of the minimum adversarial perturbation without solving any subproblem exactly. For a dual problem $D^{(j)}$, consider only having one variable $\lambda_i^{(j)}$ to be the optimization variable and fixing all the remaining variables zero; then the optimal closed-form solution for this

[1] If the precondition holds, then any point in a small ball around the center $\boldsymbol{\delta} = \boldsymbol{x}_j^- - \boldsymbol{z}$ will be a feasible solution, which implies strong duality by Slater's condition.

simplified dual QP problem is

$$\begin{aligned}\lambda_i^{(j)} &= \max\left(0, -\frac{b_i}{\|\boldsymbol{a}_i\|^2}\right),\\ D^{(j)}([0,\ldots,0,\lambda_i^{(j)},0,\ldots,0]) &= \frac{\max(-b_i,0)^2}{2\|\boldsymbol{a}_i\|^2}.\end{aligned} \tag{16.6}$$

Since $b_i = (\|\boldsymbol{z}-\boldsymbol{x}_i^+\|^2 - \|\boldsymbol{z}-\boldsymbol{x}_j^-\|^2)/2$ and $\boldsymbol{a}_i = \boldsymbol{x}_j^- - \boldsymbol{x}_i^+$ can be easily computed, we can use (16.5) to obtain a guaranteed lower bound ϵ^* by:

$$\min_{j\in[n^-]}\max_{i\in[n^+]} \frac{\max(\|\boldsymbol{z}-\boldsymbol{x}_j^-\|^2 - \|\boldsymbol{z}-\boldsymbol{x}_i^+\|^2, 0)}{2\|\boldsymbol{x}_j^- - \boldsymbol{x}_i^+\|}. \tag{16.7}$$

In addition, the certified bound in the above formulation has an interesting geometrical meaning. The inner value captures the distance between $\boldsymbol{z}$ and the bisection between $\boldsymbol{x}_i^+$ and $\boldsymbol{x}_j^-$. Perturbing $\boldsymbol{z}$ to this bisection is the smallest perturbation to make it closer to $\boldsymbol{x}_j^-$ than $\boldsymbol{x}_i^+$. If we want to perturb $\boldsymbol{z}$ so that the nearest neighbor is $\boldsymbol{x}_j^-$, then we need to make sure $\boldsymbol{z}$ is closer to $\boldsymbol{x}_j^-$ than any data point with positive label, so we take the max operation among all the distances to bisections. And to make $\boldsymbol{z}$ being classified as the negative label, we only need to make it to be closer to one of the $\boldsymbol{x}_j^-$ than positive samples, so we take the min outside. In general, we can also get improved lower bounds by optimizing more coordinates rather than one variable for each subproblem, which could lead to tighter bound but may require additional computation.

Efficient algorithms for computing 1-NN robustness

To compute the exact robustness of a 1-NN classifier, we need to solve all the primal problems (16.3), or equivalently, the corresponding dual problems (16.4). Although they are polynomial time solvable, in practice a naive algorithm is still too slow since there are $O(n)$ quadratic problems involved and each has $O(n)$ dual variables. Typically solving each QP takes $O(n^2)$, so roughly $O(n^3)$ time is required. This is too expensive when n is large.

To resolve this problem, Wang et al. (2019c) developed a method to efficiently solve the $O(n)$ subproblems. The main idea is to remove unnecessary variables and unnecessary subproblems by exploiting the primal-dual relationships. Intuitively, the final solution is the minimum among all the subproblem solutions, so if we know that the lower bound of the solution of a subproblem is larger than the current solution, then the subproblem

can be removed. As shown in the previous two sections, based on duality, *any* dual solution can give a lower bound of primal solution, so we can remove the subproblem when we found a dual solution with objective function value larger than the current minimum. Therefore, we can solve all the subproblems simutaneously and progressively remove subproblems.

On the other hand, for each subproblem, we can also try to remove unnecessary dual variables (corresponding to primal constraints). As each constraint corresponds to another point $\boldsymbol{x}_i$ where we hope the point after perturbation $(\boldsymbol{z}+\boldsymbol{\delta})$ is closer to the target point $\boldsymbol{x}_j$ than to $\boldsymbol{x}_i$, intuitively, if the point is far away, then we can remove the constraint. This can be also done by a screening algorithm derived from the primal dual relationship, and the interested readers can check more details in (Wang et al., 2019c). In general, how to efficiently compute the minimum adversarial perturbation is still an unresolved problem.

Extending beyond 1-NN

Next, we extend the above-mentioned robustness evaluation approach to K-NN models with $K > 1$. Consider the binary classification case, we let $\mathbb{S}^+ = \{\boldsymbol{x}_1^+, \ldots, \boldsymbol{x}_{n^+}^+\}$ be the set of samples with the same label with $\boldsymbol{z}$ and $\mathbb{S}^- = \{\boldsymbol{x}_1^-, \ldots, \boldsymbol{x}_{n^-}^-\}$ be the set of samples with opposite label to $\boldsymbol{z}$. We assume K is an odd number to avoid the tie in K-NN voting and both n^+, n^- are large enough. We can list all the possible combinations of $(\mathbb{I}, \mathbb{J})$ with $\mathbb{I} \subseteq [n^+]$, $\mathbb{J} \subseteq [n^-]$, $|\mathbb{I}| = (K-1)/2$, and $|\mathbb{J}| = (K+1)/2$, and then solve the following QP problem to force $\boldsymbol{z}+\boldsymbol{\delta}$ to be closer to $\boldsymbol{x}_j^-$ for all $j \in [\mathbb{J}]$ than to all instances in $\mathbb{S}^+$ except $\boldsymbol{x}_i^+$ for all $i \in [\mathbb{I}]$:

$$\begin{aligned} \epsilon^{(\mathbb{I},\mathbb{J})} = \min_{\boldsymbol{\delta}} \; & \frac{1}{2}\boldsymbol{\delta}^T\boldsymbol{\delta} \\ \text{s.t. } & \|\boldsymbol{z}+\boldsymbol{\delta}-\boldsymbol{x}_j^-\|^2 \le \|\boldsymbol{z}+\boldsymbol{\delta}-\boldsymbol{x}_i^+\|^2 \\ & \forall i \in [n^+]-\mathbb{I}, \;\; \forall j \in \mathbb{J}. \end{aligned} \tag{16.8}$$

Then the norm of the minimum adversarial perturbation can be computed by $\epsilon^* = \min_{(\mathbb{I},\mathbb{J})} \sqrt{2\epsilon^{(\mathbb{I},\mathbb{J})}}$. There will be exponential number of QPs in totat, so it is usually impossible to compute the minimum adversarial perturbation by iterating all QPs. However, we can still obtain upper and lower bounds corresponding to attack and verification. For an upper bound (attack), we can just choose a small number of $(\mathbb{I}, \mathbb{J})$ tuples and only solve these QPs. The quality of the bound will rely on how good is the selected subset of QPs.

For a lower bound (verification), by extending (16.7) to the $K > 1$ case we have the following lemma.

Lemma 3. *For a K-NN model with odd K, we have the following lower bound for the minimum adversarial perturbation:*

$$\epsilon^* \geq s\text{th}\min_{j\in[n^-]} \left(s\text{th}\max_{i\in[n^+]} \sqrt{2\epsilon^{(i,j)}} \right), \tag{16.9}$$

where $s = (K+1)/2$ *is a positive integer, "*sth min*" and "*sth max*" select the sth minimum value and the sth maximum value, respectively, and*

$$\epsilon^{(i,j)} = \frac{\max(\|\boldsymbol{z} - \boldsymbol{x}_j^-\|^2 - \|\boldsymbol{z} - \boldsymbol{x}_i^+\|^2, 0)^2}{8\|\boldsymbol{x}_j^- - \boldsymbol{x}_i^+\|^2}.$$

This is a simple extension of the 1-NN case. Note that the lower bound (16.9) can be efficiently computed. It requires $O(n^2 d)$ time to compute all $\epsilon^{(i,j)}$, and then the top-s selection can be done in linear time, so the overall complexity is just $O(n^2 d)$, which is independent of K.

Robustness of KNN vs neural network on simple problems

Using the above-mentioned algorithms implemented by Wang et al. (2019c), we are able to 1) efficiently compute the exact robustness of 1-NN models for a reasonable number of examples (e.g., for MNIST dataset, Wang et al. (2019c) showed that we can conduct evaluation in few seconds) and 2) for K-NN models, compute the certified robustness (verification) efficiently using Lemma 3.

With these algorithms, we can then compare the robustness between K-NN models and neural networks. We compare *certified robust errors*, defined as the fractions with lower bounds less than the given threshold (if an instance is wrongly classified, then the lower bound is 0), of 1-NN, a simple convolutional network (ConvNet), and a strong ℓ_2 certified defense network (RandSmooth) (Cohen et al., 2019) in Fig. 16.2 (see Chapter 13 for more details on randomized smoothing). The results show that 1-NN can achieve better certified robust errors than neural networks on these two datasets. This is partially because the proposed verification algorithm provides very tight certified regions for NN classifiers. Note that this is not saying that K-NN is more robust than neural networks on more complex datasets such as CIFAR and ImageNet. However, this is showing K-NN could obtain better certified robust errors on some simpler tasks.

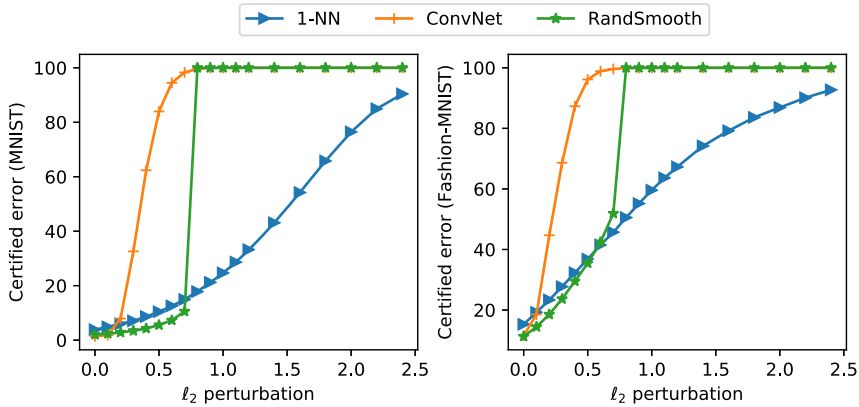

Figure 16.2 Comparing certified (robust) errors of 1-NN, ConvNet (a simple convolutional network), and RandSmooth (a defense neural network via random smoothing proposed by Cohen et al. (2019)). For RandSmooth, we choose the noise standard deviation $\sigma = 0.2$. 1-NN is more robust on these two datasets.

16.2 Defenses with nearest-neighbor classifiers

We have shown how to compute the robustness of K-NN, but can we use these techniques to improve the robustness of K-NN? The answer is affirmative.

Note that in the original K-NN models defined in the Euclidean space, we measure the distance between two points by $\|\boldsymbol{x} - \boldsymbol{x}'\|_2$. However, we can generalize the distance measurement to a family of Mahalanobis distances as follows:

$$d_M(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x} - \boldsymbol{x}')^\top M(\boldsymbol{x} - \boldsymbol{x}'), \tag{16.10}$$

where $M \in \mathbb{R}^{D \times D}$ is a positive semidefinite matrix. We can then try to learn a positive semidefinite matrix M to minimize the certified robust training error.

For a classifier f on an instance $(\boldsymbol{x}, y)$, following the previous sections, we can define the *minimal adversarial perturbation* as

$$\operatorname*{argmin}_{\boldsymbol{\delta}} \|\boldsymbol{\delta}\| \quad \text{s.t.}\, f(\boldsymbol{x} + \boldsymbol{\delta}) \neq y, \tag{16.11}$$

which is the smallest perturbation that could lead to "misclassification". Note that if $(\boldsymbol{x}, y)$ is not correctly classified, the minimal adversarial perturbation is the zero vector $\mathbf{0}$. Let $\boldsymbol{\delta}^*(\boldsymbol{x}, y)$ denote the optimal solution,

and let $\epsilon^*(\boldsymbol{x}, y) = \|\boldsymbol{\delta}^*(\boldsymbol{x}, y)\|$ be the optimal value. Since the Mahalanobis K-NN classifier is parameterized by a positive semidefinite matrix M and the training set $\mathbb{S}$, we further let the optimal solution $\boldsymbol{\delta}^*_{\mathbb{S}}(\boldsymbol{x}, y; M)$ and the optimal value $\epsilon^*_{\mathbb{S}}(\boldsymbol{x}, y; M)$ explicitly indicate their dependence on M and $\mathbb{S}$.

To learn an M that leads to robust K-NN, Wang et al. (2020c) define the objective function as

$$\min_{G \in \mathbb{R}^{D \times D}} \quad \frac{1}{N} \sum_{i=1}^{N} \ell\left(\epsilon^*_{\mathbb{S}-\{(\boldsymbol{x}_i, y_i)\}}(\boldsymbol{x}_i, y_i; M)\right) \quad \text{s.t. } M = G^\top G, \tag{16.12}$$

where $\ell : \mathbb{R} \to \mathbb{R}$ is a nonincreasing function, e.g., the hinge loss $[1 - \epsilon]_+$, exponential loss $\exp(-\epsilon)$, logistic loss $\log(1 + \exp(-\epsilon))$, or "negative" loss $-\epsilon$. The constraint $M = G^\top G$ is used to enforce the positive semidefiniteness of the M matrix, otherwise the Mahalanobis distance defined by M will not be a valid distance metric. Also, the minimal adversarial perturbation is defined on the training set excluding $(\boldsymbol{x}_i, y_i)$ itself, since otherwise a 1-nearest neighbor classifier with any distance measurement will have 100% accuracy. Therefore, the above objectie can be viewed as minimizing the "leave-one-out" certified robust error.

As mentioned in the previous section, computing the exact $\epsilon^*_{\mathbb{S}}(\boldsymbol{x}, y; M)$ is too expensive and becomes impossible when $K > 1$. Therefore we focus on improving the *certified robust error*. Let $\underline{\epsilon}^*(\boldsymbol{x}, y)$ be a lower bound of the norm of the minimal adversarial perturbation $\epsilon^*(\boldsymbol{x}, y)$, which can be computed by extending our K-NN certified bound derived in Lemma 3 to the Mahalanobis distance setting. Since the derivation is quite simple, we directly show the results below. The detailed proof can be found in (Wang et al., 2020c).

Theorem 4 (Robustness verification for Mahalanobis K-NN). *Given a Mahalanobis K-NN classifier parameterized by a neighbor parameter K, a training dataset $\mathbb{S}$, and a positive semidefinite matrix M, for any instance $(\boldsymbol{x}_{test}, y_{test})$, we have*

$$\epsilon^*(\boldsymbol{x}_{test}, y_{test}; M) \geq \underset{j: y_j \neq y_{test}}{k\text{th}\min} \; \underset{i: y_i = y_{test}}{k\text{th}\max} \; \tilde{\epsilon}(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{x}_{test}; M), \tag{16.13}$$

where kth max and kth min select the kth maximum and kth minimum, respectively, with $k = (K + 1)/2$, and

$$\tilde{\epsilon}(\boldsymbol{x}^+, \boldsymbol{x}^-, \boldsymbol{x}; M) = \frac{d_M(\boldsymbol{x}, \boldsymbol{x}^-) - d_M(\boldsymbol{x}, \boldsymbol{x}^+)}{2\sqrt{(\boldsymbol{x}^+ - \boldsymbol{x}^-)^\top M^\top M (\boldsymbol{x}^+ - \boldsymbol{x}^-)}}. \tag{16.14}$$

Note that Theorem 4 is a generalization of Lemma 3 to the Mahalanobis distance setting.

By replacing the ϵ^* in (16.12) with the lower bound derived in Theorem 4 we get a trainable objective function for adversarially robust metric learning:

$$\min_{G\in\mathbb{R}^{D\times D}} \frac{1}{N}\sum_{t=1}^{N} \ell\left(k\text{th}\min_{j:y_j\neq y_t} k\ \text{th}\max_{i:i\neq t, y_i=y_t} \tilde{\epsilon}(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{x}_t; M)\right) \quad \text{s.t. } M = G^\top G. \tag{16.15}$$

Although Eq. (16.15) is trainable since $\tilde{\epsilon}$ is a function of M, for large datasets, it is time-consuming to run the inner min-max procedure. Furthermore, since what we really care is the generalization performance of the learned metric instead of the leave-one-out robust training error, it is unnecessary to compute the exact solution. Therefore, instead of computing the kth max and kth min exactly, Wang et al. (2020c) proposed to sample positive and negative instances from the neighborhood of each training instance, which leads to the following formulation:

$$\min_{G\in\mathbb{R}^{D\times D}} \frac{1}{N}\sum_{i=1}^{N} \ell\left(\tilde{\epsilon}\left(\text{randnear}_M^+(\boldsymbol{x}_i), \text{randnear}_M^-(\boldsymbol{x}_i), \boldsymbol{x}_i; M\right)\right) \quad \text{s.t. } M = G^\top G, \tag{16.16}$$

where $\text{randnear}_M^+(\cdot)$ denotes a sampling procedure for an instance in the same class within $\boldsymbol{x}_i$'s neighborhood, and $\text{randnear}_M^-(\cdot)$ denotes a sampling procedure for an instance in a different class, also within a neighborhood of $\boldsymbol{x}_i$, and the distances are measured by the Mahalanobis distance d_M. Wang et al. (2020c) showed that sampling instances from a fixed number of nearest instances is sufficient. As a result, the optimization formulation (16.16) approximately minimizes the certified robust error and significantly improves computational efficiency.

The overall training algorithm is as follows. At every iteration, G is updated with the gradient of the loss function, whereas the calculations of $\text{randnear}_M^+(\cdot)$ and $\text{randnear}_M^-(\cdot)$ do not contribute to the gradient for efficient and stable computation. The algorithm is called *adversarially robust metric learning* (ARML).

Clearly, ARML can significantly improve the robustness of KNN models. In Table 16.1, we show that in 1-NN, when the exact robustness can be computed by the method mentioned in the previous section, ARML

Table 16.1 Certified robust errors of Mahalanobis 1-NN. The best (minimum) certified robust errors among all methods are in bold. Note that the certified robust errors of 1-NN are also the optimal empirical robust errors (attack errors), and these robust errors at the radius 0 are also the clean errors.

	ℓ_2-radius	0.000	0.500	1.000	1.500	2.000	2.500
MNIST	Euclidean	0.033	0.112	0.274	0.521	0.788	0.945
	NCA	0.025	0.140	0.452	0.839	0.977	1.000
	LMNN	0.032	0.641	0.999	1.000	1.000	1.000
	ITML	0.073	0.571	0.928	1.000	1.000	1.000
	LFDA	0.152	1.000	1.000	1.000	1.000	1.000
	ARML (Ours)	**0.024**	**0.089**	**0.222**	**0.455**	**0.757**	**0.924**
	ℓ_2-radius	0.000	0.500	1.000	1.500	2.000	2.500
Fashion-MNIST	Euclidean	0.145	0.381	0.606	0.790	0.879	0.943
	NCA	**0.116**	0.538	0.834	0.950	0.998	1.000
	LMNN	0.142	0.756	0.991	1.000	1.000	1.000
	ITML	0.163	0.672	0.929	0.998	1.000	1.000
	LFDA	0.211	1.000	1.000	1.000	1.000	1.000
	ARML (Ours)	0.127	**0.348**	**0.568**	**0.763**	**0.859**	**0.928**
	ℓ_2-radius	0.000	0.100	0.200	0.300	0.400	0.500
Splice	Euclidean	0.320	0.513	0.677	0.800	0.854	0.880
	NCA	**0.130**	0.252	0.404	0.584	0.733	0.836
	LMNN	0.190	0.345	0.533	0.697	0.814	0.874
	ITML	0.306	0.488	0.679	0.809	0.862	0.882
	LFDA	0.264	0.434	0.605	0.760	0.845	0.872
	ARML (Ours)	**0.130**	**0.233**	**0.370**	**0.526**	**0.652**	**0.758**
	ℓ_2-radius	0.000	0.100	0.200	0.300	0.400	0.500
Pendigits	Euclidean	0.032	0.119	0.347	0.606	0.829	0.969
	NCA	0.034	0.202	0.586	0.911	0.997	1.000
	LMNN	0.029	0.183	0.570	0.912	0.995	0.999
	ITML	0.049	0.308	0.794	0.991	1.000	1.000
	LFDA	0.042	0.236	0.603	0.912	0.998	1.000
	ARML (Ours)	**0.028**	**0.115**	**0.344**	**0.598**	**0.823**	**0.967**
	ℓ_2-radius	0.000	0.150	0.300	0.450	0.600	0.750
Satimage	Euclidean	0.108	0.642	0.864	0.905	0.928	0.951
	NCA	0.103	0.710	0.885	0.915	0.940	0.963
	LMNN	**0.092**	0.665	0.871	0.912	0.944	0.969
	ITML	0.127	0.807	0.979	1.000	1.000	1.000
	LFDA	0.125	0.836	0.919	0.956	0.992	1.000
	ARML (Ours)	0.095	**0.605**	**0.839**	**0.899**	**0.920**	**0.946**

continued on next page

Table 16.1 (*continued*)

	ℓ_2-radius	0.000	0.500	1.000	1.500	2.000	2.500
USPS	Euclidean	0.045	0.224	0.585	0.864	**0.970**	**0.999**
	NCA	0.056	0.384	0.888	0.987	1.000	1.000
	LMNN	0.046	0.825	1.000	1.000	1.000	1.000
	ITML	0.060	0.720	0.999	1.000	1.000	1.000
	LFDA	0.098	1.000	1.000	1.000	1.000	1.000
	ARML	**0.043**	**0.204**	**0.565**	**0.857**	**0.970**	**0.999**

can consistently improve the robustness over 1-NN model with the original Euclidean norm. Note that in addition to Euclidean (original Euclidean space 1-NN model) and ARML (adversarially robust metric learning), in the comparison, we also include some traditional metric learning baselines including NCA (Goldberger et al., 2004), LMNN (Weinberger and Saul, 2009), ITML (Davis et al., 2007), and LFDA (Sugiyama, 2007). Note that those methods try to learn a better M metric to improve the classification performance. We can observe that the ARML algorithm can consistently achieve the best robustness.

On the other hand, when running KNN with $K > 1$, as mentioned in the previous section, it becomes computational infeasible to compute the exact robustness score. However, we can still compute certified robustness of KNN models using Theorem 4, and Table 16.2 demonstrates that ARML can consistently improve the certified error. For more empirical results, we refer the readers to (Wang et al., 2020c), who also compare the certified robustness of ARML with neural networks on some simple datasets such as MNIST and FashionMNIST.

16.3 Evaluating the robustness of decision tree ensembles

Next, we discuss the adversarial robustness of decision-tree based classifiers, including several widely used ensemble tree models such as random forest (RF) and gradient boosting decision tree (GBDT) models. These models have shown significant performance, outperforming neural networks on many data mining tasks.

Robustness of a single decision tree

First, we discuss the case when there is a single decision tree. We assume that the decision tree has n nodes and the root node is indexed as 0. For a given example $x = [x_1, \ldots, x_d]$ with d features, starting from the root node,

Table 16.2 Certified robust errors (left) and empirical robust errors (right) of Mahalanobis *K*-NN. The best (minimum) robust errors among all methods are in bold. The empirical robust errors at the radius 0 are also the clean errors.

		Certified robust errors						**Empirical robust errors**					
	ℓ_2-radius	0.000	0.500	1.000	1.500	2.000	2.500	0.000	0.500	1.000	1.500	2.000	2.500
MNIST	Euclidean	0.038	0.134	0.360	0.618	0.814	0.975	0.031	0.063	0.104	0.155	0.204	0.262
	NCA	**0.030**	0.175	0.528	0.870	0.986	1.000	**0.027**	0.063	0.120	0.216	0.330	0.535
	LMNN	0.040	0.669	1.000	1.000	1.000	1.000	0.036	0.121	0.336	0.775	0.972	1.000
	ITML	0.106	0.731	0.943	1.000	1.000	1.000	0.084	0.218	0.355	0.510	0.669	0.844
	LFDA	0.237	1.000	1.000	1.000	1.000	1.000	0.215	1.000	1.000	1.000	1.000	1.000
	ARML	0.034	**0.101**	**0.276**	**0.537**	**0.760**	**0.951**	0.032	**0.055**	**0.077**	**0.109**	**0.160**	**0.213**
	ℓ_2-radius	0.000	0.500	1.000	1.500	2.000	2.500	0.000	0.500	1.000	1.500	2.000	2.500
Fashion-MNIST	Euclidean	0.160	0.420	0.650	0.800	0.895	0.946	0.143	0.227	0.298	0.360	0.420	0.489
	NCA	**0.144**	0.557	0.832	0.946	1.000	1.000	**0.121**	0.232	0.343	0.483	0.624	0.780
	LMNN	0.158	0.792	0.991	1.000	1.000	1.000	0.140	0.364	0.572	0.846	0.983	0.999
	ITML	0.236	0.784	0.949	1.000	1.000	1.000	0.209	0.460	0.692	0.892	0.978	1.000
	LFDA	0.291	1.000	1.000	1.000	1.000	1.000	0.263	0.870	0.951	0.975	0.988	0.995
	ARML	0.152	**0.371**	**0.589**	**0.755**	**0.856**	**0.924**	0.134	**0.202**	**0.274**	**0.344**	**0.403**	**0.487**
	ℓ_2-radius	0.000	0.100	0.200	0.300	0.400	0.500	0.000	0.100	0.200	0.300	0.400	0.500
Splice	Euclidean	0.333	0.558	0.826	0.965	0.988	0.996	0.306	0.431	0.526	0.608	0.676	0.743
	NCA	**0.103**	**0.209**	0.415	0.659	0.824	0.921	**0.103**	**0.173**	0.274	0.414	0.570	0.684
	LMNN	0.149	0.332	0.630	0.851	0.969	0.994	0.149	0.241	0.357	0.492	0.621	0.722
	ITML	0.279	0.571	0.843	0.974	0.995	0.997	0.279	0.423	0.525	0.603	0.675	0.751
	LFDA	0.242	0.471	0.705	0.906	0.987	0.997	0.242	0.371	0.466	0.553	0.637	0.737
	ARML	0.128	0.221	**0.345**	**0.509**	**0.666**	**0.819**	0.128	0.196	**0.273**	**0.380**	**0.497**	**0.639**

continued on next page

Table 16.2 (*continued*)

		Certified robust errors						Empirical robust errors					
	ℓ_2-radius	0.000	0.100	0.200	0.300	0.400	0.500	0.000	0.100	0.200	0.300	0.400	0.500
	Euclidean	0.039	0.126	0.316	0.577	0.784	0.937	0.036	0.085	0.155	0.248	0.371	0.528
	NCA	0.038	0.196	0.607	0.884	0.997	1.000	0.038	0.103	0.246	0.428	0.637	0.804
Pendigits	LMNN	0.034	0.180	0.568	0.898	0.993	0.999	**0.030**	0.096	0.246	0.462	0.681	0.862
	ITML	0.060	0.334	0.773	0.987	1.000	1.000	0.060	0.149	0.343	0.616	0.814	0.926
	LFDA	0.047	0.228	0.595	0.904	1.000	1.000	0.043	0.104	0.248	0.490	0.705	0.842
	ARML	**0.035**	**0.114**	**0.308**	**0.568**	**0.780**	**0.937**	0.034	**0.078**	**0.138**	**0.235**	**0.368**	**0.516**
	ℓ_2-radius	0.000	0.150	0.300	0.450	0.600	0.750	0.000	0.150	0.300	0.450	0.600	0.750
	Euclidean	**0.101**	0.579	0.842	0.899	0.927	0.948	0.091	0.237	0.482	0.682	**0.816**	0.897
	NCA	0.117	0.670	0.886	0.915	0.936	0.961	0.101	0.297	0.564	0.746	0.876	0.931
Satimage	LMNN	0.105	0.613	0.855	0.914	0.944	0.961	**0.090**	0.269	0.548	0.737	0.855	0.910
	ITML	0.130	0.768	0.959	1.000	1.000	1.000	0.109	0.411	0.757	0.939	0.990	1.000
	LFDA	0.128	0.779	0.904	0.958	0.995	1.000	0.112	0.389	0.673	0.860	0.950	0.986
	ARML	0.103	**0.540**	**0.824**	**0.898**	**0.920**	**0.943**	0.092	**0.228**	**0.464**	**0.668**	0.817	**0.896**
	ℓ_2-radius	0.000	0.500	1.000	1.500	2.000	2.500	0.000	0.500	1.000	1.500	2.000	2.500
	Euclidean	0.063	0.239	0.586	0.888	0.977	1.000	0.058	0.125	0.211	0.365	0.612	**0.751**
	NCA	0.072	0.367	0.903	0.986	1.000	1.000	0.063	0.158	0.365	0.686	0.899	0.980
USPS	LMNN	0.062	0.856	1.000	1.000	1.000	1.000	0.055	0.359	0.890	0.999	1.000	1.000
	ITML	0.082	0.696	0.999	1.000	1.000	1.000	0.072	0.273	0.708	0.987	1.000	1.000
	LFDA	0.134	1.000	1.000	1.000	1.000	1.000	0.118	0.996	1.000	1.000	1.000	1.000
	ARML	**0.057**	**0.203**	**0.527**	**0.867**	**0.971**	**0.997**	**0.053**	**0.118**	**0.209**	**0.344**	**0.572**	0.785

x traverses the decision tree model until reaching a leaf node. Each internal node, say node i, has two children and a univariate feature-threshold pair (t_i, η_i) to determine the traversal direction: x will be passed to the left child if $x_{t_i} \leq \eta_i$ and to the right child otherwise. Each leaf node has a value v_i corresponding to the predicted class label for a classification tree or a real value for a regression tree.

Similar to the previous sections, we first discuss how to compute the minimum adversarial perturbation that can change the decision of a given test sample. More specifically, for a given tree-based model f, a test sample x, and the corresponding label y_0, the goal is to find

$$r^* = \min_{\delta} \|\delta\|_\infty \quad \text{s.t.} \quad f(x+\delta) \neq y_0. \tag{16.17}$$

If f is a single decision tree, there is a simple linear-time algorithm for computing r^*. The main idea is to compute a d-dimensional box for each leaf node such that any example in this box will fall into this leaf. Mathematically, the box of node i is defined as the Cartesian product $B^i = (l_1^i, r_1^i] \times \cdots \times (l_d^i, r_d^i]$ of d intervals on the real line. By definition the root node has the box $[-\infty, \infty] \times \cdots \times [-\infty, \infty]$, and given the box of an internal node i, its children's boxes can be obtained by changing only one interval of the box based on the split condition (t_i, η_i). More specifically, if p, q are left and right child nodes of node i, respectively, then we set their boxes $B^p = (l_1^p, r_1^p] \times \cdots \times (l_d^p, r_d^p]$ and $B^q = (l_1^q, r_1^q] \times \cdots \times (l_d^q, r_d^q]$ by

$$(l_t^p, r_t^p] = \begin{cases} (l_t^i, r_t^i] & \text{if } t \neq t_i, \\ (l_t^i, \min\{r_t^i, \eta_i\}] & \text{if } t = t_i, \end{cases} \quad (l_t^q, r_t^q] = \begin{cases} (l_t^i, r_t^i] & \text{if } t \neq t_i, \\ (\max\{l_t^i, \eta_i\}, r_t^i] & \text{if } t = t_i. \end{cases} \tag{16.18}$$

After computing the boxes for internal nodes, we can also obtain the boxes for leaf nodes using (16.18). Therefore computing the boxes for all the leaf nodes of a decision tree can be done by a traversal (either depth-first or breadth-first) of the tree with time complexity $O(nd)$.

This technique can be easily generalized to the ℓ_p norm perturbations with $p < \infty$. In particular, to certify whether there exist any misclassified points under perturbation $\|\delta\|_p \leq \epsilon$, we can enumerate boxes for all n leaf nodes and check the minimum distance from $\boldsymbol{x}_0$ to each box. The following proposition shows that the ℓ_p norm distance between a point and a box can be computed in $O(d)$ time, and thus the complete robustness verification problem for a single tree can be solved in $O(dn)$ time.

Theorem 5. *Given a box* $B = (l_1, r_1] \times \cdots \times (l_d, r_d]$ *and a point* $\boldsymbol{x} \in \mathbb{R}^d$*, the minimum* ℓ_p *distance* ($p \in [0, \infty]$) *from* x *to* B *is* $\|z - x\|_p$*, where*

$$z_i = \begin{cases} x_i, & l_i \leq x_i \leq u_i, \\ l_i, & x_i < l_i, \\ u_i, & x_i > u_i. \end{cases} \tag{16.19}$$

We can use this theorem to compute the minimum ℓ_p norm perturbation to move the test sample x to the box. The minimum ℓ_p norm adversarial perturbation can then be easily computed by enumerating the possibilities to move x to each leaf node with an opposite label.

Robustness of ensemble decision stumps

A decision stump is a decision tree with only one root node and two leaf nodes. We assume that there are T decision stumps and the ith decision stump gives the prediction

$$f^i(x) = \begin{cases} w_l^i & \text{if } x_{t_i} < \eta^i, \\ w_r^i & \text{if } x_{t_i} \geq \eta^i. \end{cases}$$

An ensemble of decision stumps has been widely used in boosting models when using decision stump as the base classifier, such as Adaboost (Freund and Schapire, 1997).

The robustness of ensemble decision stumps have been studied by Wang et al. (2020e) and Andriushchenko and Hein (2019). We have shown in the previous subsection that the robustness evaluation can be done in linear time for any ℓ_p norms, but interestingly, the complexity will varies for different p when considering robustness evaluation of ensemble decision stumps.

The prediction of a decision stump ensemble $F(x) = \sum_i f^i(x)$ can be decomposed into each feature in the following way. For each feature j, letting $j_1, \ldots, j_{T_j}$ be the decision stumps using feature j, we can collect all the thresholds $[\eta^{j_1}, \ldots, \eta^{j_{T_j}}]$. Without loss of generality, assume that $\eta^{j_1} \leq \cdots \leq \eta^{j_{T_j}}$. Then the prediction values assigned in each interval can be denoted as

$$g^j(x_j) = v^{j_t} \quad \text{if } \eta^{j_t} < x_j \leq \eta^{j_{t+1}}, \tag{16.20}$$

where

$$v^{j_t} = w_l^{j_1} + \cdots + w_l^{j_t} + w_r^{j_{t+1}} + \cdots + w_r^{j_{T_j}},$$

and x_j is the value of sample x on feature j. The overall prediction can be written as the sum over the predicted values of each feature,

$$F(x) = \sum_{j=1}^{d} g^j(x_j), \tag{16.21}$$

and the final prediction is given by $y = \text{sgn}(F(x))$.

ℓ_∞ and ℓ_0 norm robustness evaluation. For ℓ_∞ norm robustness evaluation, we try to study whether there exists a perturbation δ within an ϵ-ℓ_∞ ball around a certain input x that can change the prediction. As the prediction can be decomposed into each dimension, we just need to investigate within ϵ perturbation what is the worst-case perturbation for each dimension. This can be trivially evaluated with a linear scan, so the evaluation can be done in linear time. Mathematically, for each feature j, we want to know *the maximum decrease of prediction value by changing this feature,* which can be computed as

$$c^j = \min_{t:(\eta^{j_{t-1}}, \eta^{j_t}] \cap [x_j - \epsilon, x_j + \epsilon] \neq \emptyset} v^{j_t} - g^j(x_j). \tag{16.22}$$

Then the minimum prediction that can be achieved within the ϵ ℓ_∞ ball will be $\sum_j c^j$.

On the other hand, when considering the ℓ_0 norm ball perturbation set, it is equivalent to saying that we can perturb at most ϵ features (assuming that ϵ is an integer), and the perturbation on each feature is not constrained. Therefore, the optimal perturbation will be choosing the features with the bottom-ϵ lowest c^j values to perturb, so the computation can be done also in linear time.

Other ℓ_p norm robustness evaluation

The difficulty of ℓ_p norm robustness verification is that the perturbations on each feature are correlated, so we cannot separate all the features as in (Andriushchenko and Hein, 2019) for the ℓ_∞ norm case. In the following, we prove that the complete ℓ_p norm verification is NP-complete by showing a reduction from Knapsack to ℓ_p norm ensemble stump verification. This shows that ℓ_p norm verification can belong to a different complexity class compared to the ℓ_∞ norm case.

Theorem 6. *Solving ℓ_p norm robustness verification (with soundness and completeness) for an ensemble decision stumps is NP-complete when $p \in (0, \infty)$.*

As computing the exact robustness is NP-Complete, Wang et al. (2020e) proposed several ways to find an upper bound for robustness verification,

based on the connection between ℓ_p norm robustness evaluation and the Kanpsack problem. More details can be found in (Wang et al., 2020e).

Robustness of ensemble decision trees

Now we discuss the robustness verification for tree (instead of decision stump) ensembles. Assuming that the tree ensemble has K decision trees, we use $S^{(k)}$ to denote the set of leaf nodes of tree k and $m^{(k)}(x)$ to de note the function that maps the input example x to the leaf node of tree k according to the traversal rule of the tree. Given an input example x, the tree ensemble will pass x to each of these K trees independently and reach K leaf nodes $i^{(k)} = m^{(k)}(x)$ for $k = 1, \ldots, K$. Each leaf node will assign a prediction value $v_{i^{(k)}}$. For simplicity, we start with the binary classification case, with original label of x being $y_0 = -1$ and the goal (for adversary) is to turn the label into $+1$. For binary classification, the prediction of the tree ensemble is computed by $\text{sign}(\sum_{k=1}^{K} v_{i^{(k)}})$, which covers both GBDTs and random forests, two widely used tree ensemble models. Since x is assumed to have label $y_0 = -1$, we know $\text{sign}(\sum_k v_{i^{(k)}}) < 0$ for x, and our task is to verify whether the sign of the sum can be flipped within $\text{Ball}(x, \epsilon)$.

We consider the decision problem of robustness verification to verify whether there exists an adversarial perturbation within a small region. A naive analysis will need to check all the points in $\text{Ball}(x, \epsilon)$, which is uncountably infinite. To reduce the search space to a finite one, the main idea is to represent the space by all the possible choices of leaf nodes of these K trees. More specifically, we let $C = \{(i^{(1)}, \ldots, i^{(K)}) \mid i^{(k)} \in S^{(k)} \ \forall k = 1, \ldots, L\}$ to be all the possible tuples of leaf nodes, and let $C(x) = [m^{(1)}(x), \ldots, m^{(K)}(x)]$ be the function that maps x to the corresponding leaf nodes. Therefore a tuple C directly determines the model prediction $\sum v_C := \sum_k v_{i^{(k)}}$, and the good thing is that this space only have finite (although exponentially many) of choices.

Now we define a valid tuple for robustness verification.

Definition 1. *A tuple $C = (i^{(1)}, \ldots, i^{(K)})$ is called valid if there exists $x' \in \text{Ball}(x, \epsilon)$ such that $C = C(x')$.*

The decision problem of robustness verification can then be written as

$$\textit{Does there exist a valid tuple } C \textit{ such that } \sum v_C > 0?$$

Next, we show how to model the set of valid tuples. We have two observations. First, if a tuple contains any node i where the box i is out of the

range from x ($\inf_{x' \in B^i}\{\|x - x'\|_\infty\} > \epsilon$), then it is invalid. Second, since the box corresponding to each leaf node captures all the possible points fall into the node, there exists x such that $C = C(x)$ if and only if the intersection of all these boxes are nonempty. This can be written as $B^{i^{(1)}} \cap \cdots \cap B^{i^{(K)}} \neq \emptyset$.

Based on these observations, we can represent the set of valid tuples as cliques in a graph $G = (V, E)$, where each node is a valid leaf node $V := \{i | B^i \cap \text{Ball}(x, \epsilon) \neq \emptyset\}$, and each edge indicates that two end nodes have non-empty intersection $E := \{(i, j) | B^i \cap B^j \neq \emptyset\}$. The graph will then be a K-partite graph since there cannot be any edge between nodes from the same tree, and thus maximum cliques in this graph will have K nodes. We denote each part of the K-partite graph as V_k. Here a "part" means a disjoint and independent set in the K-partite graph. The following lemma shows that intersections of boxes have very nice properties.

Lemma 4. *For boxes $B^1, \ldots, B^K$, if $B^i \cap B^j \neq \emptyset$ for all $i, j \in [K]$, let $\bar{B} = B^1 \cap B^2 \cap \cdots \cap B^K$ be their intersection. Then $\bar{B}$ is also a box, and $\bar{B} \neq \emptyset$.*

Details of the proof can be found in Chen et al. (2019b). Therefore, each K-clique (fully connected subgraph with K nodes) in G can be viewed as a set of leaf nodes that has nonempty intersection with each other and also has nonempty intersection with $\text{Ball}(x, \epsilon)$, so the intersection of those K boxes and $\text{Ball}(x, \epsilon)$ will be a nonempty box, which implies that each K-clique corresponds to a valid tuple of leaf nodes.

Lemma 5. *A tuple $C = (i^{(1)}, \ldots, i^{(K)})$ is valid if and only if nodes $i^{(1)}, \ldots, i^{(K)}$ form a K-clique (maximum clique) in graph G constructed above.*

Therefore the robustness verification problem can be formulated as

$$\textit{Is there a maximum clique } C \textit{ in } G \textit{ such that } \sum v_C > 0? \tag{16.23}$$

This reformulation indicates that the tree ensemble verification problem can be solved by an efficient maximum clique enumeration algorithm. Unfortunately, it has been shown that enumerating all the maximum cliques is NP-complete and takes $O(3^{\frac{m}{3}})$ time, which is time consuming, and therefore an efficient alternative is proposed by Chen et al. (2019b), which we will briefly mention below.

An efficient multilevel algorithm for robustness verification of decision trees

Fig. 16.3 illustrates the multilevel algorithm proposed by Chen et al. (2019b). There are four trees, and each tree has four leaf nodes. A node is

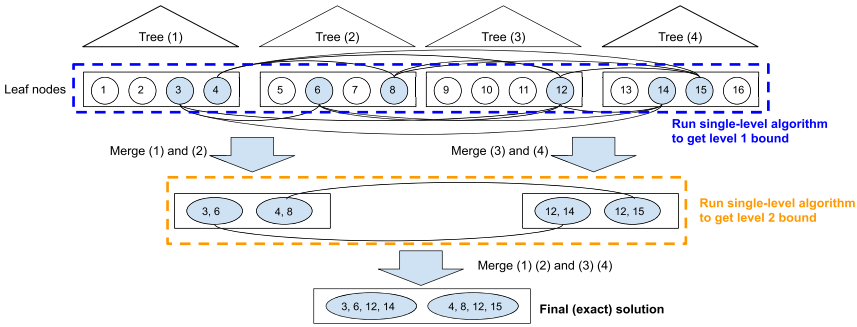

Figure 16.3 The multilevel verification algorithm proposed in Chen et al. (2019b). Lines between leaf node i on tree t_1 and leaf node j on t_2 indicate that their ℓ_∞ feature boxes intersect (i.e., there exists an input such that tree 1 predicts v_i and tree 2 predicts v_j).

colored if it has nonempty intersection with Ball(x, ϵ); uncolored nodes are discarded. To answer question (16.23), we need to compute the maximum $\sum v_C$ among all K-cliques, denoted by v^*. As mentioned before, for robustness verification, we only need to compute an upper bound of v^* to get a lower bound of minimal adversarial perturbation. In the following, we will first discuss algorithms for computing an upper bound at the top level and then show how our multiscale algorithm iteratively refines this bound until reaching the exact solution v^*.

Bounds for a single level

For each single level, Chen et al. (2019b) proposed a simple approach to compute an upper bound of the maximimum clique value. This is done by assuming the graph is fully connected between any two parts (two independent sets) of the level. By making this assumption, we will only increase the maximum clique value as we are introducing strictly more edges. With this assumption, the maximum sum of node values is the sum of the maximum value of each part (independent set). In Fig. 16.3, if we focus on one of the layers, this can be simply computed by finding the maximum value in each group and sum them up.

Another slightly better approach is exploiting the edge information but only between trees t and $t+1$. If we search over all the length-K paths $[i^{(1)}, \ldots, i^{(K)}]$ from the first to the last part and define the value of a path to be $\sum_k v_{i^{(k)}}$, then the maximum-valued path will be a valid upper bound. This can be computed in linear time using dynamic programming by running from the first part to the last part and keep track of the current

maximum value on each node in the current part. This dynamic programming algorithm costs slightly more time but can get a much tighter upper bound.

Merging T independent sets

To refine the relatively loose single-level bound, the multilevel algorithm partitions the graph into K/T subgraphs, each with T independent sets. Within each subgraph, we find all the T-cliques and use a new "pseudo-node" to represent each T-clique. T-cliques in a subgraph can be enumerated efficiently if we choose T to be a relatively small number (e.g., 2 or 3 in the experiments).

Next, we need to form the graph at the next level by merging nodes in the previous level. By Lemma 4 we know that the intersection of T boxes will still be a box, so each T-clique is still a box and can be represented as a pseudo-node in the level-2 graph. Also, because each pseudo-node is still a box, we can easily form edges between pseudo-nodes to indicate the nonempty overlapping between them, and this will be a (K/T)-partite boxicity graph since no edge can be formed for the cliques within the same subgraph. Thus we get the level-2 graph. With the level-2 graph, we can again run the single-level algorithm to compute an upper bound on v^* to get a lower bound of r^*, but differently from the level-1 graph, now we already considered all the within-subgraph edges, so the bounds will be much tighter.

The overall multilevel framework

We can run the algorithm level-by-level, and at each level the single-level algorithm will output a valid upper bound for verification. As we go to the next level, since the interconnections between the merged parts will be considered, the bounds will also get tighter. In the final level, the pseudo-nodes will correspond to the K-cliques in the original graph, and the maximum value will be the exact solution for the max clique problem. Therefore the algorithm can be viewed as an anytime algorithm that refines the upper bound level-by-level until reaching the maximum value.

Training robust tree ensembles

Similar to neural networks, to enhance the robustness of tree ensembles, we can consider both empirical defense and certified defense. For both of

them, the goal is to train a tree to minimize the robust loss defined as

$$\min_f E_{(x,y)\sim D} \max_{x' \in B(x,\epsilon)} \ell(f(x), y), \tag{16.24}$$

where for simplicity, we assume that $B(x, \epsilon)$ is the ℓ_∞ ball $B(x, \epsilon) = \{x' : \|x' - x\|_\infty \leq \epsilon\}$. In decision tree training, a commonly used technique is the top-down approach, where at each step, we determine how to split the node or, equivalently, how to pick the feature and threshold on a leaf node by minimizing the objective function. Therefore, to train the robust trees, we need to determine the best split of the node based on the robust loss (16.24) instead of the regular loss ((16.24) without the inner max). Since the training of decision trees is often complicated and there exists multiple training methods, we omit the detailed training algorithms here. Instead, we will just describe the algorithms conceptually and direct the interested readers to corresponding papers.

To minimize the adversarial loss, we can consider both the upper bound or the lower bound (similarly to adversarial training versus certified training in robust neural network learning). Chen et al. (2019a) proposed the first robust tree training algorithm based on a lower bound of robust loss. Conceptually, when determining how to split a node, a simple but effective way is testing all the possible split features and thresholds, computing the (robust) loss of each choice, and then choosing the best one. The naive procedure to compute the objective for all the splits requires $O(N^2D)$ time, as there are D features, each with N thresholds, and the objective function involves N samples. However, there exists an efficient way to test all the thresholds for a single feature. With the original nonrobust loss, since after ordering the thresholds from left to right, when knowing the objective function for a particular threshold, we can update it to the objective function value for the next threshold in $O(1)$ time. This leads to $O(ND)$ time complexity for each split. However, when considering the robust loss, since each sample can be either $+\epsilon$ or $-\epsilon$ depending on the worst-case scenario, it becomes nontrivial to update the objective function from a threshold to the next one in $O(1)$ time. To overcome this challenge, Chen et al. (2019a) only considers a particular assignment for the perturbation instead of the worst-case one, leading to an algorithm minimizing the lower bound of robust loss in each split.

On the other hand, it is also possible to do the split based on the upper bound of robust error computed by robust verification methods mentioned above. In particular, Andriushchenko and Hein (2019) applied their verification method, and Wang et al. (2020e) generalized the work to ℓ_p norm

certified training. With their methods, it is possible to train a certified robust tree ensemble and outperform certified robust neural network in some simpler datasets (such as MNIST). Although it is not realistic to use tree-based methods for large image classification problems such as ImageNet, tree-based models are used in many data mining tasks and have been the main component for reranking in the retrieval systems, so it is still important to train a robust tree for those problems.