


Detecting Evasion Attacks in Deployed Tree Ensembles

Laurens Devos^[0000-0002-1549-749X], Lorenzo Perini^[0000-0002-5929-9727],
Wannes Meert^[0000-0001-9560-3872], and Jesse Davis^[0000-0002-3748-9263]

KU Leuven, Leuven, Belgium
`firstname.lastname@kuleuven.be`

Abstract. Tree ensembles are powerful models that are widely used. However, they are susceptible to evasion attacks where an adversary purposely constructs an adversarial example in order to elicit a misprediction from the model. This can degrade performance and erode a user’s trust in the model. Typically, approaches try to alleviate this problem by verifying how robust a learned ensemble is or robustifying the learning process. We take an alternative approach and attempt to detect adversarial examples in a post-deployment setting. We present a novel method for this task that works by analyzing an unseen example’s output configuration, which is the set of leaves activated by the example in the ensemble’s constituent trees. Our approach works with any additive tree ensemble and does not require training a separate model. We evaluate our approach on three different tree ensemble learners. We empirically show that our method is currently the best adversarial detection method for tree ensembles.

Keywords: Evasion attack detection · Tree ensembles

1 Introduction

Tree ensembles such as (gradient) boosted trees and random forests are a popular class of models. However, like many other model families, such as neural networks [2, 18, 40], they are susceptible to **evasion attacks** [9, 12, 14, 23, 47, 50]. In this attack setting, a previously trained model is deployed and, while operating in the wild, is exposed to **adversarial examples** that an adversary purposely constructed to elicit a misprediction from the model. Such examples are undesirable because they degrade a model’s performance and erode a user’s trust in the model. For tree ensembles, the literature attempts to deal with evasion attacks in one of two ways. First, verification techniques attempt to ascertain how robust a learned ensemble is to adversarial examples [9, 12, 34] by empirically determining how much an example would have to be perturbed (according to some norm) for its predicted label to change. Second, the problem can be addressed at training time by trying to learn a more robust model by adding adversarial examples to the training set [23], pruning the training data [49], changing aspects of the learner such as the splitting criteria [1, 7, 8, 44] or the objective [21], relabeling

the values in the leaves [45], using constraint solvers to learn optimal trees [46], or interleaving learning and verification [35].

This paper explores an alternative approach to mitigating the effect of an evasion attack. Given a current example for which a prediction is required, we attempt to ascertain if this current example is adversarial or not. If the example is identified as being adversarial, then the deployed model could refrain from making a prediction similar to a learning with rejection setting [11]. While this question has been extensively explored for neural networks, this is not the case for tree ensembles. Unfortunately, most existing methods for neural networks are not applicable to tree ensembles because they use properties unique to neural networks [51]. For example, some modify the model [17, 19, 30], learn other models (e.g., nearest neighbors) on top of the network’s intermediate representations [16, 24, 27, 39], or learn other models on top of the gradients [37]. Moreover, nearly all methods focus only on detecting evasion attacks for image classification.

Tree ensembles are powerful because they combine the predictions made by many trees. Hence, the prediction procedure involves sorting the given example to a leaf node in each tree. The ordered set of the reached leaf nodes is an *output configuration* of the ensemble and fully determines the ensemble’s resulting prediction. However, there are many more possible output configurations than there are examples in the data used to train the model. For example, the California housing dataset [31] only has eight attributes, but training an XGBoost ensemble containing 6, 7, or 8 trees each of at most depth 5 yields 62 248, 173 826, and 385 214 output configurations respectively.¹ These numbers (far) exceed the 20,600 examples in the dataset. The situation will be worse for the larger ensembles sizes that are used in practice. Our hypothesis is that in an evasion attack

adversarial examples exploit **unusual output configurations**, that is, ones that are very different to those observed in the data used to train the model.

That is, small, but carefully selected perturbations can yield an example that is quite similar to another example observed during training, but yields an output configuration that is far away from those covered by the training data.

Based on this intuition, we present a novel method to detect an evasion attack based on assessing whether an example encountered post deployment has an unusual output configuration. When a new example is encountered, our approach encodes it by its output configuration and then measures the distance between the encoded example and its nearest (encoded) neighbor in a reference set. If this distance is sufficiently high, the example is flagged as being an adversarial one and the model can abstain from making a prediction. Our approach has several benefits. First, it is general: it works with any additive tree ensemble. Second, it is integrated: it does not require training a separate model to identify adversarial examples, one simply has to set a threshold on the distance. Finally, it

¹ Computed using Veritas [12].

is surprisingly fast as the considered distance metric can be efficiently computed by exploiting instruction level parallelism (SIMD).

Empirically, we evaluate and compare our approach on three ensemble methods: gradient boosted trees (XGBoost [10]), random forests [4], and GROOT [44], which is an approach for training robust tree ensembles. We empirically show that our method outperforms multiple competing approaches for detecting adversarial examples post deployment for all three considered tree ensembles. Moreover, it can detect adversarial examples with a comparable computational effort.

2 Preliminaries

We assume a d -dimensional input space $\mathcal{X} = \mathbb{R}^d$ and a target space $\mathcal{Y} = \{0, 1\}$. Though we evaluate our algorithm on binary classification problems, the method generalizes to multi-classification and regression. The random variable X with distribution $p(X)$ represents a d dimensional feature vector, and the random variable Y with distribution $p(Y)$ represents the target variable. The instances $(x, y) \in D$ are sampled from the joint distribution $p(X, Y)$, written $(x, y) \sim p(X, Y)$. We use A_k to denote the k th attribute in the data, $k = 1, \dots, d$.

Additive Tree Ensembles. This paper proposes a method that works with *additive tree ensembles of decision trees*. These are a frequently used family of machine learning models encompassing both random forests and (gradient) boosted trees. Excellent open-source packages are available such as XGBoost [10], LightGBM [25], Scikit-learn [32] and many others. The models are learned from a dataset $D \subseteq \mathcal{X} \times \mathcal{Y}$ and define a mapping from \mathcal{X} to \mathcal{Y} .

A binary decision tree T is a recursive data structure consisting of nodes. It starts at a root node that is not a descendant of any other node. Every node is either a leaf node storing an output value, or an internal node storing a test (e.g., *is attribute A_k less than 5?*) and two child nodes. A tree is evaluated for an example $x \in \mathcal{X}$ starting at the root node. If the node is an internal node, the test is executed for the node and, if successful, x moves down to the left child node, or if unsuccessful, moves down to the right child node, and the procedure recurs. If the node is a leaf, the output value of the leaf is the prediction for x .

An additive ensemble of trees \mathbf{T} is a sum of M trees. The prediction is the sum of the predictions of the trees: $\mathbf{T}(x) = \sigma(\sum_{m=1}^M T_m(x))$. The transformation σ depends on the ensemble type and the learning task. Figure 1 shows an example of a tree ensemble.

Output Configuration (OC). The output configuration of an example x in an ensemble \mathbf{T} is the ordered set of leaves $(l^{(1)}, \dots, l^{(M)})$ visited by x in each tree of the ensemble. Each leaf is reached by exactly one root-to-leaf path. The split conditions in the internal nodes along the root-to-leaf paths of all leaves in an OC define the *box* in the input space of the OC. That is, for an OC $o \in \mathcal{O}$, $\text{box}(o) = \prod_{k \leq d} [u_k, v_k)$, where each $[u_k, v_k)$ is an interval constraining the k th attribute. Note that a valid OC has a non-empty box. This also means that

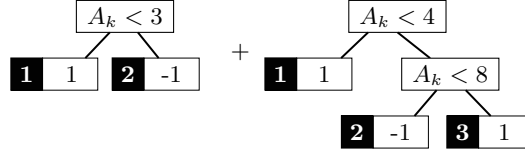


Fig. 1. A simple tree ensemble. Each leaf has an identifier in black, and leaf class predictions.

the *OC-space* is not just the Cartesian product of all leaves because some leaf combinations are impossible. For example, in Figure 1, (1, 2) and (1, 3) are invalid OCs because A_k cannot be both less than 3 and greater than 4 at the same time.

We define an operator $\text{OC} : x \mapsto (l^{(1)}, \dots, l^{(M)})$ mapping an example x to its output configuration, where $l^{(m)}$ are leaf identifiers. This naturally defines the **OC-space** $\mathcal{O} = \{\text{OC}(x) \mid x \in \mathcal{X}\}$ as the discrete space of all OCs. There is a one-to-one map between the boxes and the *OC-space* and the set of all boxes is a partition of the input space. Moreover, if two examples have the same OC, they belong to the same box and the ensemble \mathbf{T} produces the same output for both examples.

Evasion Attacks and Adversarial Examples Adversaries can attack machine learning models at training time (*poisoning*), or at deployment time (*evasion*) [3]. We focus on detecting *evasion attacks*. An evasion attack consists of constructing an example \tilde{x} close to a real correctly labeled example x such that \tilde{x} elicits a misprediction, i.e., $\mathbf{T}(x) \neq \mathbf{T}(\tilde{x})$ and $\|\tilde{x} - x\| < \varepsilon$ for some norm $\|\cdot\|$ and some $\varepsilon > 0$. We call such an \tilde{x} an **adversarial example**. The same definition of an adversarial example is also used in previous work on tree ensembles [9, 12, 23], and corresponds to the *prediction-change* setting of [13].

Evasion attacks can vary on two important dimensions: (1) black-box vs. white-box, and (2) low-confidence vs. high-confidence. A white-box method has full access to the structure of the model, while a black-box method only uses the model’s predictions. Low- and high-confidence refers to how the attacker tries to manipulate the confidence that the ensemble assigns to the incorrect label. For example, a low-confidence adversarial example in binary classification problem has a predicted probability close to 0.5.

3 Detecting Evasion Attacks with *OC-score*

We assume a post-deployment setting where a tree ensemble \mathbf{T} is operating in the wild. The OC-score method then solves the following task:

- Given:** a deployed tree ensemble \mathbf{T} and an unseen example $x \in \mathcal{X}$ for which a prediction is required,
Do: assign a score $\text{OC-score}(x)$ to x indicating whether x is an adversarial example generated by an evasion attack.

Our algorithm is based on the fact that for sufficiently large models, **the vast majority of the model’s possible OCs will not be observed in the data used to train the model.**

Decision tree learners employ heuristics to select splitting criteria in internal nodes with the goal of separating examples from different classes. Consequently, most leaf nodes tend to be (strongly) predictive of one class. Correctly classified examples will tend to have OCs consisting largely of leaves that are predictive of the correct class. An evasion attack constructs an adversarial example using carefully selected perturbations to a small number of attribute values so that enough leaves in the original example’s OC are replaced by leaves that are predictive of the opposing class, yielding *unusual OCs* with an incorrect output label. This suggests that measuring how unusual an OC is, i.e., measuring how similar a newly encountered example’s OC is to those that appear in the training set, is an effective strategy to detect adversarial examples.

3.1 The *OC-score* Metric

Our approach requires a learned ensemble \mathbf{T} and a reference set $D_R \subseteq D$ of correctly classified examples. It constructs two reference sets $R_{\hat{y}} = \{\text{OC}(x) \mid (x, y) \in D_R, y = \hat{y}\}$, one for each class, by encoding the examples in D_R into the *OC-space* by finding each one’s output configuration.

Given a newly encountered example $x \in \mathcal{X}$, the ensemble is used to obtain its predicted label $\hat{y} = \mathbf{T}(x)$ and output configuration $o = \text{OC}(x)$. Then it receives an *OC-score* by computing the Hamming distance to the closest OC in $R_{\hat{y}}$:

$$\text{OC-score}(x) = \min_{o' \in R_{\hat{y}}} h(o, o'), \quad (1)$$

with $h(o, o') = \sum_{m=1}^M \mathbf{1}[l^{(m)} \neq l'^{(m)}]$ the number of leaves that differ between the OCs and $l^{(m)}$ and $l'^{(m)}$ the m th leaf in o and o' respectively. Higher *OC-scores* correspond to a higher chance of being an adversarial example. Operationally, a threshold can be set on the *OC-scores* to flag potential adversarial examples: when the threshold is exceeded, the model abstains from making a prediction.

The *OC-score* algorithm can be implemented very efficiently by exploiting instruction-level parallelism using SIMD. The reference set’s OCs can be compactly represented as a matrix of small integers (e.g., 8-bit integers), with in the i th row the identifiers of the leaves in the OC of the i th example in R . To compute $\text{OC-score}(x)$, we slide the vector $\text{OC}(x)$ over the rows of this matrix and compute the Hamming distance. This can be done on x86 using the 256-bit AVX* SIMD extensions. The full details are in the supplement and the source code is available at <https://github.com/laudv/ocscore>.

3.2 Theoretical Analysis

Our approach flags adversarial examples generated as part of an evasion attack by setting a threshold on the *OC-score*. It will only be accurate if, on average,

the expected *OC-score* of an adversarial example is larger than the expected *OC-score* of a normal example. We prove that with two reasonable assumptions that this is indeed the case.

The first is the split-uniqueness assumption, which requires that no two splits in the ensemble are identical. The second assumption requires that adversarial examples produce OCs that are unlikely to be observed in the data used to train the model. Therefore, we first need to formally define how likely an OC is and what constitutes an unlikely OC.

Definition 1 (Probability of an OC). *Let X be a random variable denoting the feature vector, the probability $\mathbb{P}_X(o)$ of an OC $o \in \mathcal{O}$ is the probability of finding a normal example $x \sim p(X)$ in the box of o :*

$$\mathbb{P}_X(o) = \mathbb{P}_X(\text{box}(o)) = \mathbb{P}(X \in \text{box}(o)). \quad (2)$$

We can then restrict ourselves to low probability OCs:

Definition 2 (Unlikely OC). *Given an $\varepsilon > 0$, an OC $o \in \mathcal{O}$ is unlikely if $\mathbb{P}_X(o) < \varepsilon$.*

The supplement shows that there always exists an ε such that unlikely OCs exist.

We want the *OC-score* to be high for adversarial examples, and low for normal ones. This leads to the definition of an unusual OC. The *OC-score* measures the distance to the closest reference set example in *OC-space*. For the *OC-score* to be large with high probability, the likelihood of finding a reference set example in the *OC-space* neighborhood must be low. Specifically, given an unlikely OC o_ε and its *OC-space* neighbors o' , $h(o_\varepsilon, o') \leq w$ for some distance $w \leq M$, the likelihood of finding a reference set example $x \sim p(X)$ in the a box of any neighbor must be low.

Definition 3 (Unusual OCs). *Choose ε such that unlikely OCs exist. Then, for any distance $w \leq M$, an unlikely OC o_ε is unusual if its *OC-space* neighborhood is less likely to contain a reference set example than the neighborhood of a non-unlikely OC o :*

$$\mathbb{P}_X\left(\bigcup\{\text{box}(o') \mid h(o', o_\varepsilon) \leq w, o' \in \mathcal{O}\}\right) \leq \mathbb{P}_X\left(\bigcup\{\text{box}(o') \mid h(o', o) \leq w, o' \in \mathcal{O}\}\right) \quad (3)$$

If we assume that adversarial examples use unusual OCs, then we prove that adversarial examples must have larger expected *OC-scores* on average than normal examples:

Theorem 1. *Let $\varepsilon > 0$. Let $\mathcal{O}_{<\varepsilon} = \{o \in \mathcal{O} \mid \mathbb{P}_X(o) < \varepsilon\}$ the set of unlikely OCs and $\mathcal{O}_{\geq\varepsilon} = \mathcal{O} \setminus \mathcal{O}_{<\varepsilon}$. Let $x_\varepsilon \sim p(X|\mathcal{O}_{<\varepsilon})$, with $p(X|\mathcal{O}_{<\varepsilon})$ the distribution of instances with unlikely OCs, i.e., $x_\varepsilon \sim p(X)$ such that $x_\varepsilon \in \text{box}(o_\varepsilon)$ and $o_\varepsilon \in \mathcal{O}_{<\varepsilon}$. Similarly, let $x \sim p(X|\mathcal{O}_{\geq\varepsilon})$, where $p(X|\mathcal{O}_{\geq\varepsilon})$ is the distribution of instances with non-unlikely OCs. Then,*

$$\mathbb{E}_{x \sim p(X|\mathcal{O}_{\geq\varepsilon})} [\mathbb{E}_R[\text{OC-score}(x)]] < \mathbb{E}_{x_\varepsilon \sim p(X|\mathcal{O}_{<\varepsilon})} [\mathbb{E}_R[\text{OC-score}(x_\varepsilon)]] \quad (4)$$

The proof is in the supplement.

4 Related Work

Beyond the approaches mentioned in the introduction for detecting adversarial examples in neural networks, there are methods that look at the behavior of the decision boundary in an example’s neighborhood [15, 36, 42]. Unfortunately, these methods do not work well with tree ensembles because the use of binary axis-parallel splits make them step functions, which makes exploring the neighborhood difficult. Also, the work investigating the relation between model uncertainty and adversarial examples [20, 29] is relevant to this paper.

Certification methods [26, 33] try to guarantee that, given an example, no adversarial examples exist within an l_∞ ball. In practice, this is achieved by (1) certifying the training data where labels are known, or (2) sampling in the neighborhood around an unseen instance and certifying that the unseen example and the sampled instances have the same predicted label. These methods achieve tractability by relaxing and approximating the (neural) model (e.g., bounding the activation function).

Each example’s *OC-score* can be viewed as a model’s secondary output with the predicted class being its primary output. This fits into the larger task of machine learning with a reject option [11]. Rejection aims to identify test examples for which the model was not properly trained. For such examples, the model’s predictions have an elevated risk of being incorrect, and hence may not be trustworthy. An example can be rejected due to ambiguity (i.e., how well the decision boundary is defined in a region) or novelty (i.e., how anomalous an example is with respect to the observed training data) [22]. The *OC-score* metric goes beyond measuring ambiguity in an ensemble (i.e., the model’s confidence in a prediction). Therefore, it can detect adversarial examples even if they fall in a region of the input space where the model’s decision boundary appears to be well defined given the training data.

The random forest manual [5] discusses defining distances between training examples in an analogous manner to *OC-score*. Typically, (variations on) this distance has been used for tasks such as clustering [38], feature transformations [43], or making tree ensembles more interpretable [41]. To our knowledge, it has not been used for detecting adversarial examples.

5 Experimental Evaluation

Our experimental evaluation addresses three questions:

- Q1. Can *OC-score* more accurately detect adversarial examples than its competitors?
- Q2. What is each approach’s prediction time cost associated with detecting adversarial examples?
- Q3. How does the size of the reference set affect the performance of our *OC-score* metric?

We compare our *OC-score* to four approaches:

Ambiguity (*ambig*) This approach uses the intuition that because adversarial examples are somehow different than the training ones, the model will be uncertain about an adversarial example’s predicted label [20]. This entails deciding whether an example lies near a model’s decision boundary. This can be done by ranking examples according to the uncertainty of the classifier: $\text{ambig}(x) = 1 - |2p_{\mathbf{T}}(x) - 1|$, where $p_{\mathbf{T}}$ is the probability of the positive class as predicted by the ensemble \mathbf{T} for an example x .

Local outlier factor (*lof*) [6] Another intuition to detect adversarial examples is to employ an anomaly detector under the assumption that adversarial examples are drawn from a different distribution than non-adversarial ones. *lof* is a state-of-the-art unsupervised anomaly detection method that assigns a score to each example denoting how anomalous it is. This approach entails learning a *lof* model which is applied to each example.

Isolation forests (*iforest*) An isolation forest [28] is a state-of-the-art anomaly detector. It learns a tree ensemble that separates anomalous from normal data points by splitting on a randomly selected attribute using a randomly chosen split value between the minimum and maximum value of the attribute. Outliers tend to be split off earlier in the trees, so the depth of an example in the tree is indicative of how normal an example is. Again, this requires learning a separate model at training time.

ML-LOO (*mlloo*) This is an approach for detecting adversarial examples from the neural network literature [48]. Unlike most other approaches, it is model agnostic as it looks at statistics of the features. It uses the accumulated *feature attributions* to rank examples: $\text{std}_k \{p_{\mathbf{T}}(x) - p_{\mathbf{T}}(x_{(k)})\}$, where $p_{\mathbf{T}}$ is the probability prediction of ensemble \mathbf{T} , and $x_{(k)}$ is x with the k th attribute set to 0. The observation in [48] is that variation in the feature attributions is larger for adversarial examples.

Table 1. Datasets’ characteristics and learners’ hyperparameter settings. #F and n are the number of features resp. examples. M is the number of trees. The learning rate and tree depth for XGBoost are η and d_T . *calhouse* is a regression dataset converted to binary classification by predicting when the target is greater than the median value.

	#F	n	η	M	d_T	class balance
calhouse	8	20.6k	0.5	100	5	50%
electricity	8	45.3k	0.4	80	8	58%
covtype	54	581.0k	0.5	80	6	51%
higgs	33	250.0k	0.1	100	8	66%
ijcnn1	22	141.7k	0.9	50	5	90%
mnist2v4	784	13.8k	0.7	50	5	51%
fmnist2v4	784	14.0k	0.1	100	4	50%
webspam	254	350.0k	0.9	50	5	39%

5.1 Experimental Methodology

We mimic the post-deployment evasion attack setting using 5-fold cross validation. In each fold, an ensemble is trained on four folds of clean training data. Then, using random subsets (with replacement) of the remaining fold, we generate 4×500 adversarial examples using four different methods. The adversarial examples are then supplemented by 2000 randomly selected normal (i.e., unmodified) examples. The resulting test set has an equal number of adversarial examples (4×500) as normal examples (2000). The different detection methods are then evaluated on the test set by comparing their performance on the task of distinguishing the adversarial examples from the normal examples.

We test our approach on the eight benchmark datasets listed in Table 1. All datasets are min-max normalized to make perturbations of the same size to different attributes comparable. To demonstrate our approach’s generality, we consider three types of additive tree ensembles: (1) XGBoost boosted trees [10], (2) Scikit-learn random forests [32], and (3) GROOT robustified random forests [44], which modifies the criteria for selecting the split condition when learning a tree to make them more robust against adversarial examples. Due to space constraints, we only show plots for some of the datasets. The results for the remaining datasets are along the same lines and are provided in the supplement.

Experimental settings. For a given dataset, each learner has the same number of trees in the ensemble. The XGBoost trees are depth-limited to the values in Table 1. The random forests are not depth limited, but are limited to have at most 255 leaves. GROOT ensembles are limited to depth 8. Table 1 reports other characteristics for the datasets. Except for Q3, the reference set contains all correctly classified training examples. We use the scikit-learn [32] implementation for *lof* and *iforest* and use the default hyper-parameters. The supplement reports the average accuracies of the learned models on each dataset and the attack model ϵ of the GROOT ensembles. All experiments ran on an Intel E3-1225 with 32GB of memory. Multi-threading was enabled for all methods.

Simulating the Evasion Attacks We use four different evasion attack methods. Each of these generates a different set of 500 adversarial examples. All the methods use the l_∞ -norm to measure the perturbation size. They are:

- **LT-attack** (abbreviated ‘*lt*’, [50]) iteratively moves a random initial example \tilde{x} with $\mathbf{T}(\tilde{x}) \neq \mathbf{T}(x)$ towards the attacked example x in steps within a neighborhood such that $\|x - \tilde{x}\|_\infty$ is minimized.
- **Kantchelian attack** (abbr. ‘*kan*’, [23]) is an exact approach² that directly minimizes $\|x - \tilde{x}\|_\infty$.
- **Veritas attack** (abbr. ‘*ver*’, [12]) is an approximate search-based approach that optimizes the ensemble’s output in an l_∞ box of size δ centered around an original example x : $\max_{\tilde{x}} \mathbf{T}(\tilde{x})$ subject to $\|\tilde{x} - x\|_\infty < \delta$.

² We use Veritas’s binary-search approach to find the closest adversarial example because it is an order of magnitude faster than their mixed-integer linear programming solution.

- **Cube attack** (abbr. ‘*cub*’, [1]) is a model agnostic approach. Given an example x , it iteratively makes a random perturbation. If the perturbation moves the prediction towards the desired label it is accepted and otherwise it is rejected. The procedure is successful if the desired label is reached after a fixed number of iterations.

The Cube attack is the only *black box* approach. It does not use the ensemble’s inner structure and uses only the ensemble’s predictions.

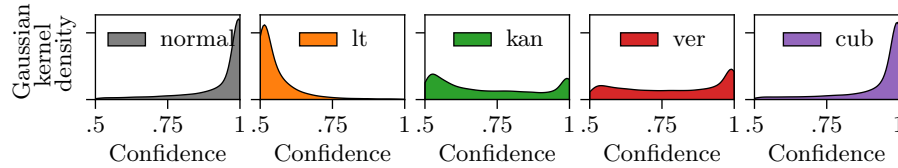


Fig. 2. The distribution of the confidence values $0.5 + |0.5 - p_{\mathbf{T}}(x)|$ of the XGBoost ensembles for the test examples and the adversarial examples of all generation methods. The results are aggregated over all datasets and all folds.

5.2 Results Q1: Detecting Evasion Attacks

The task is to distinguish the adversarial from the normal examples. Each method generates a score, ranking the examples ideally from least adversarial (low score) to most adversarial (high score).

First, we evaluate the ranking performance using the area under the ROC (ROC AUC). This measures the quality of a ranking with respect to the classification task of separating adversarial and normal examples. Table 2 shows the mean AUC values for each method and the standard deviations over the five folds for XGBoost, random forests, and GROOT. Averaged over all adversarial sets, *OC-score* outperforms all other methods. An interesting observation is that XGBoost is less robust than random forests, and random forests are less robust than GROOT ensembles. The perturbations required to flip the label become larger as the models become more robust, and thus some adversarial examples become outliers in the traditional sense. Nonetheless, the results show that even robust GROOT ensembles benefit from detecting evasion attacks during deployment.

Second, we evaluate the performance of each detection method on one important characteristic of the evasion attack: model confidence. The confidence with which the ensemble predicts the incorrect class is computed as $0.5 + |0.5 - p_{\mathbf{T}}(x)|$. The attack can simply try to elicit a misprediction or it can try to construct an adversarial example such that the model is highly confident in its misprediction. The LT- and Kantchelian attacks optimize the distance $\|x - \tilde{x}\|_{\infty}$ between the original example x and the adversarial example \tilde{x} . Hence, the methods find the smallest perturbation such that \tilde{x} just crosses the decision boundary, which often

Table 2. Average ROC AUC for each detection method on all four sets of adversarial examples on each dataset. Best results are in bold.

XGBoost								
	calhouse	electricity	covtype	higgs	ijcnn1	mnist2v4	fmnist2v4	webspam
ocscore	.92\pm.00	.94\pm.01	1.0\pm.00	.90\pm.01	.98\pm.00	.99\pm.00	.94\pm.01	.99\pm.00
ambig	.76 \pm .01	.78 \pm .01	.76 \pm .01	.82 \pm .01	.75 \pm .02	.88 \pm .02	.74 \pm .04	.96 \pm .01
iforest	.61 \pm .02	.59 \pm .02	.56 \pm .01	.60 \pm .02	.62 \pm .01	.50 \pm .00	.55 \pm .01	.51 \pm .02
lof	.67 \pm .00	.53 \pm .01	.65 \pm .01	.54 \pm .02	.73 \pm .01	.61 \pm .01	.60 \pm .01	.53 \pm .01
mlloo	.55 \pm .01	.61 \pm .02	.49 \pm .01	.72 \pm .01	.62 \pm .02	.89 \pm .02	.73 \pm .03	.93 \pm .01
Random forests								
	calhouse	electricity	covtype	higgs	ijcnn1	mnist2v4	fmnist2v4	webspam
ocscore	.92\pm.01	.97\pm.00	1.0\pm.00	.91\pm.00	.99\pm.00	1.0\pm.00	.98\pm.00	.99\pm.00
ambig	.87 \pm .01	.82 \pm .01	.88 \pm .01	.82 \pm .01	.97 \pm .00	.95 \pm .01	.77 \pm .02	.96 \pm .01
iforest	.61 \pm .02	.63 \pm .02	.60 \pm .01	.61 \pm .01	.56 \pm .01	.46 \pm .01	.61 \pm .01	.51 \pm .01
lof	.77 \pm .00	.63 \pm .01	.81 \pm .00	.60 \pm .01	.76 \pm .00	.61 \pm .00	.65 \pm .01	.59 \pm .01
mlloo	.37 \pm .02	.48 \pm .02	.39 \pm .02	.58 \pm .01	.37 \pm .03	.93 \pm .01	.51 \pm .02	.36 \pm .01
GROOT								
	calhouse	electricity	covtype	higgs	ijcnn1	mnist2v4	fmnist2v4	webspam
ocscore	.87\pm.02	.96\pm.00	.85\pm.01	.96\pm.01	.99\pm.00	.94\pm.01	.94\pm.00	.99\pm.00
ambig	.85 \pm .01	.80 \pm .01	.72 \pm .04	.82 \pm .02	.99\pm.00	.92 \pm .02	.77 \pm .03	.86 \pm .01
iforest	.67 \pm .01	.78 \pm .02	.70 \pm .03	.84 \pm .01	.77 \pm .03	.55 \pm .01	.65 \pm .01	.72 \pm .01
lof	.83 \pm .01	.73 \pm .01	.84 \pm .00	.81 \pm .01	.89 \pm .01	.69 \pm .01	.71 \pm .01	.90 \pm .01
mlloo	.37 \pm .01	.55 \pm .01	.34 \pm .02	.52 \pm .02	.04 \pm .01	.86 \pm .01	.66 \pm .03	.46 \pm .04

Table 3. The average ROC AUC values for each detection method on *low-confidence* (*lt* and *kan*) and *high-confidence* (*ver* and *cub*) adversarial examples for XGBoost (XGB), random forests (RF), and GROOT (GRT). The best results are in bold.

	low confidence			high confidence		
	XGB	RF	GRT	XGB	RF	GRT
ocscore	.94 \pm .05	.98 \pm .02	.93 \pm .10	.97\pm.03	.95\pm.05	.93\pm.07
ambig	.98\pm.02	.99\pm.01	.98\pm.03	.63 \pm .21	.76 \pm .15	.70 \pm .16
iforest	.53 \pm .02	.52 \pm .05	.64 \pm .10	.60 \pm .11	.63 \pm .15	.77 \pm .13
lof	.52 \pm .04	.57 \pm .08	.70 \pm .12	.69 \pm .18	.78 \pm .18	.90 \pm .10
mlloo	.80 \pm .19	.54 \pm .18	.52 \pm .26	.58 \pm .21	.45 \pm .25	.43 \pm .26

results in the misprediction having a low confidence. Veritas directly optimizes the model’s output, so it specifically looks for the example with the maximally incorrect output within the l_∞ -box. Cube uses a fixed number of iterations and attempts to improve the confidence at each step. Because of the random model-agnostic approach, many iterations are required, resulting in high-confidence adversarial examples.

Figure 2 shows the distribution of ensemble’s confidences for each set of adversarial examples and the normal test set examples, averaged over all datasets for XGBoost. As expected, the LT- and Kantchelian attacks produce low-confidence adversarial examples, and the Veritas and Cube attacks produce high-confidence ones. Additionally, the figure shows that XGBoost has very confident predictions for normal examples. This has the immediate effect that low-confidence attacks are weaker attack vectors that are easily detectable by *ambiguity*. *OC-score* also performs extremely well on these (see Table 3). However, *ambiguity* performs poorly on the more challenging high-confidence adversarial examples. Only *OC-score* consistently works well for high-confidence adversarial examples.

Next, we further investigate the effect of model confidence on detection performance for XGBoost. We order the examples by the ensemble’s confidence and compute each method’s detection accuracy within a window. Regardless of the window, a method’s global median score over all examples is used to make a hard prediction. This is a sensible choice because the test set contains 50% normal and 50% adversarial examples. The top plots in Figure 3 show the accuracy and the bottom plots show the model’s average confidence for the examples in the considered window. The x -axis shows the fraction of examples processed as the window slides from left (low confidence) to right (high confidence). *OC-score* offers consistently strong performance across the full range of confidences and outperforms the other methods apart from *ambiguity* on the windows with the lowest average confidence. *ambiguity* offers good performance for low confidence examples. However, its detection performances declines dramatically when the x -axis approaches 0.5 because these windows start to contain roughly equal numbers of adversarial and normal examples. Its performance only rebounds slightly thereafter because it is incapable of detecting high-confidence adversarial examples. *lof* performs poorly on low confidence windows but can perform better in the highest confidence because a (small) fraction of these adversarial examples are outliers wrt the training data. ML-LOO (apart from *mnist2v4*) and *iforest* perform poorly regardless of the window.

The anomaly detectors (*iforest* and *lof*) tend to perform poorly in most settings. *lof* performs reasonably well for the Cube-attack examples on *calhouse*, *covtype*, *ijcnn1* and *mnist2v4*. This might be because this is the crudest black-box generation method whose examples might be closer to *out of distribution* than actually being truly deceitful adversarial examples (see per-dataset results in the supplement). ML-LOO has highly variable performance and is consistently worse than *OC-score*. It tends to work best on image data. However, it frequently performs worse than random on several other datasets (e.g., *calhouse*, *covtype*). For the XGBoost ensembles, ML-LOO is reasonably effective at detecting the

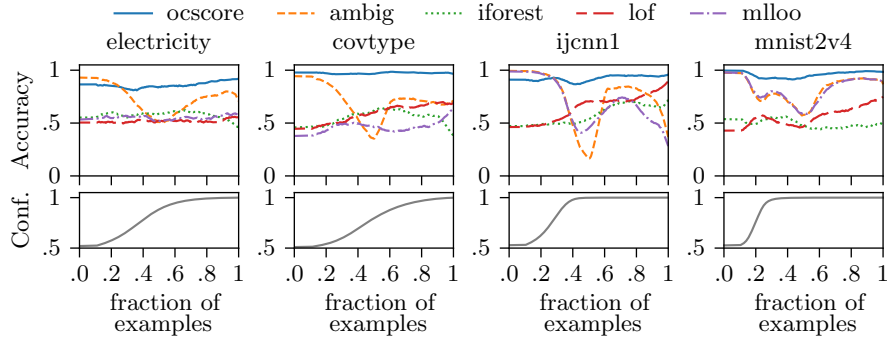


Fig. 3. Relation between detection performance and prediction confidence for XGBoost by sliding fixed-sized window over the full set of test examples (normal and the four sets of adversarial examples) sorted by model confidence. Top: Accuracy for each detection method in the window. Bottom: Average confidence in the window. The x -axis is the fraction of examples already considered by a window.

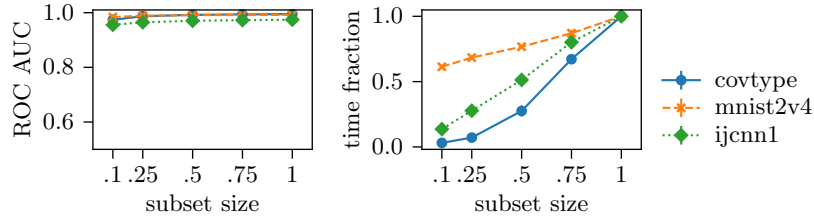


Fig. 4. Left: OC-score’s ROC AUC values for detecting adversarial vs. non-adversarial test examples as a function of the reference set size. Right: fraction of time used to compute the OC-score relative to the case the full training set is used. Using a small reference set has a minimal impact on OC-score’s performance.

low-confidence adversarial examples, but its performance is poor in general for high-confidence ones.

5.3 Results Q2: Prediction Time Cost

Regardless of the detection method, whether an example is adversarial can usually be computed in well under 1 millisecond (see timing results in the supplement). *ambiguity* consistently takes less than 0.01ms because it is a simple mathematical computation. *iforest* is also fast (< 0.15 ms) because it only requires executing a tree ensemble. *OC-score* takes less than 0.2ms for 6 out of 8 datasets. Its prediction time scales with the number of trees and the reference set size, so higher prediction times are measure for the larger datasets. However, the results in Subsection 5.4 indicate that it is possible to decrease

the size of the reference set without degrading performance. ML-LOO’s costs comes from computing an importance for each feature. Hence, it yields longer times for datasets with many features (1.4ms for *mnist2v4*, 2.6ms for *fmnist2v4*, < 0.5 ms otherwise). The evaluation time of *lof* is similar for all larger datasets (< 0.14 ms) because we limit the training set size to a random subset of at most 20 000 examples (affects *covtype*, *higgs*, *ijcnn1*, and *webspam*) because using the full training set takes hours.

5.4 Results Q3: Size of the Reference Set

Finally, we explore the effect of the reference set size on the detection performance. Reference set examples are randomly sampled from the set of correctly classified training examples (see model accuracies in supplement). Figure 4 shows the results for this experiment on three datasets using XGBoost. The plot on the left shows ROC AUC values for detecting adversarial vs. non-adversarial examples. These values are stable for all datasets. There is a small decline in performance for the smallest reference set proportion, where the number of examples in the reference set ranges from 1050 for *fmnist2v4* to 41 700 for *covtype*.

The plot on the right shows the time to compute the *OC-scores* as the reference set size is varied relative to using the full reference set. For *ijcnn1*, the relative time reduction follows the relative subset sizes almost exactly. The results for *covtype* are even better. With barely any effect on the detection performance using only 10% of the full reference set, the evaluation time drops by 97%. We expect that this is due to CPU cache performance (e.g. fewer cache misses). *mnist2v4* also sees a considerable reduction in time, but not as impressive as the previous two datasets. We suspect this is due to the smaller reference set, and the relatively higher constant overhead. *OC-score* already is really fast on *mnist2v4*, however, taking only 0.03ms per example using the full reference set.

These experiments suggest that using a small reference set drastically improves the evaluation time without degrading performance. Note that changing the reference set does not require relearning the underlying ensemble used to make predictions, which is still learned using the full training set.

6 Conclusions and Discussion

This paper explored how to detect evasion attacks for tree ensembles. Our approach works with any additive tree ensemble and does not require training a separate model. If a newly encountered example’s output configuration differs substantial from those in the reference set, then it is more likely to be adversarial. Empirically, our *OC-score* metric resulted in superior detection performance.

Acknowledgements. This research is supported by the Research Foundation – Flanders (LD: 1SB1322N; LP: 1166222N), the Flemish Government under the “Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen” program (JD), the European Union’s Horizon Europe Research and Innovation program under the grant agreement TUPLES No. 101070149 (JD), and KU Leuven Research Fund (JD: iBOF/21/075; WM: IOF).

Ethical Statement

Machine learning is widely used in many different application areas. With the wide adoption, machine learned models, including tree ensembles, increasingly become high-stake targets for attackers who might employ evasion attacks to achieve their goal. This work proposes a defense method against evasion attacks for tree ensembles. Together with other approaches like robust tree ensembles, this work is a step forward in our ability to protect against evasion attacks. This could further improve the trust in machine learning, and further accelerate its adoption, especially in sensitive application areas.

Improved defenses will likely also result in the development of improved counter-attacks. We strongly feel that it is in the interest of the research community that (1) the research community stays on top of these developments so that machine learning libraries can adapt if necessary, and (2) all work done in this area is open-access. For that reason, all resources and codes are publicly available at <https://github.com/laudv/ocscore>.

References

1. Andriushchenko, M., Hein, M.: Provably robust boosted decision stumps and trees against adversarial attacks. In: *Advances in Neural Information Processing Systems*. vol. 32 (2019)
2. Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., Roli, F.: Evasion attacks against machine learning at test time. In: *Machine Learning and Knowledge Discovery in Databases*. pp. 387–402 (2013)
3. Biggio, B., Roli, F.: Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition* **84**, 317–331 (2018)
4. Breiman, L.: Random forests. *Machine learning* **45**(1), 5–32 (2001)
5. Breiman, L., Cutler, A.: Random forests manual. <https://www.stat.berkeley.edu/~breiman/RandomForests> (2002)
6. Breunig, M.M., Kriegel, H.P., Ng, R.T., Sander, J.: Lof: identifying density-based local outliers. In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. pp. 93–104 (2000)
7. Calzavara, S., Lucchese, C., Tolomei, G., Abebe, S.A., Orlando, S.: Treant: training evasion-aware decision trees. *Data Mining and Knowledge Discovery* **34**(5), 1390–1420 (2020)
8. Chen, H., Zhang, H., Boning, D., Hsieh, C.J.: Robust decision trees against adversarial examples. In: *Proceedings of the 36th International Conference on Machine Learning*. pp. 1122–1131 (2019)
9. Chen, H., Zhang, H., Si, S., Li, Y., Boning, D., Hsieh, C.J.: Robustness verification of tree-based models. In: *Advances in Neural Information Processing Systems*. vol. 32, pp. 12317–12328 (2019)
10. Chen, T., Guestrin, C.: XGBoost: A scalable tree boosting system. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. p. 785–794. KDD ’16 (2016)
11. Cortes, C., DeSalvo, G., Mohri, M.: Learning with rejection. In: *Proceedings of The 27th International Conference on Algorithmic Learning Theory (ALT 2016)* (2016)

12. Devos, L., Meert, W., Davis, J.: Versatile verification of tree ensembles. In: Proceedings of the 38th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 139, pp. 2654–2664 (2021)
13. Diochnos, D., Mahloujifar, S., Mahmoody, M.: Adversarial risk and robustness: General definitions and implications for the uniform distribution. *Advances in Neural Information Processing Systems* **31** (2018)
14. Einziger, G., Goldstein, M., Sa’ar, Y., Segall, I.: Verifying robustness of gradient boosted models. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 2446–2453 (2019)
15. Fawzi, A., Fawzi, H., Fawzi, O.: Adversarial vulnerability for any classifier. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. pp. 1186–1195 (2018)
16. Feinman, R., Curtin, R.R., Shintre, S., Gardner, A.B.: Detecting adversarial samples from artifacts. arXiv preprint arXiv:1703.00410 (2017)
17. Gong, Z., Wang, W., Ku, W.S.: Adversarial and clean data are not twins. arXiv preprint arXiv:1704.04960 (2017)
18. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: Proceedings of the 3rd International Conference on Learning Representations (2015)
19. Grosse, K., Manoharan, P., Papernot, N., Backes, M., McDaniel, P.: On the (statistical) detection of adversarial examples. arXiv preprint arXiv:1702.06280 (2017)
20. Grosse, K., Pfaff, D., Smith, M.T., Backes, M.: The limitations of model uncertainty in adversarial settings. In: 4th workshop on Bayesian Deep Learning (NeurIPS 2019) (2018)
21. Guo, J.Q., Teng, M.Z., Gao, W., Zhou, Z.H.: Fast provably robust decision trees and boosting. In: Proceedings of the 39th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 162, pp. 8127–8144 (2022)
22. Hendrickx, K., Perini, L., der Plas, D.V., Meert, W., Davis, J.: Machine learning with a reject option: A survey. *CoRR* **abs/2107.11277** (2021)
23. Kantchelian, A., Tygar, J.D., Joseph, A.: Evasion and hardening of tree ensemble classifiers. In: Proceedings of the 33rd International Conference on Machine Learning. pp. 2387–2396. PMLR (2016)
24. Katzir, Z., Elovici, Y.: Detecting adversarial perturbations through spatial behavior in activation spaces. In: 2019 International Joint Conference on Neural Networks. pp. 1–9 (2019)
25. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.Y.: LightGBM: A highly efficient gradient boosting decision tree. In: Advances in Neural Information Processing Systems. vol. 30, pp. 3146–3154 (2017)
26. Lecuyer, M., Atlidakis, V., Geambasu, R., Hsu, D., Jana, S.: Certified robustness to adversarial examples with differential privacy. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 656–672. IEEE (2019)
27. Lee, K., Lee, K., Lee, H., Shin, J.: A simple unified framework for detecting out-of-distribution samples and adversarial attacks. *Advances in Neural Information Processing Systems* **31** (2018)
28. Liu, F.T., Ting, K.M., Zhou, Z.H.: Isolation forest. In: Proceedings of the 8th IEEE International Conference on Data Mining. pp. 413–422. IEEE (2008)
29. Liu, X., Li, Y., Wu, C., Hsieh, C.J.: Adv-BNN: Improved adversarial defense through robust Bayesian neural network. In: Proceedings of the 7th International Conference on Learning Representations (2019), <https://openreview.net/forum?id=rk4Qso0cKm>

30. Metzen, J.H., Genewein, T., Fischer, V., Bischoff, B.: On detecting adversarial perturbations. In: *Proceedings of 5th International Conference on Learning Representations* (2017)
31. Pace, R.K., Barry, R.: Sparse spatial autoregressions. *Statistics & Probability Letters* **33**(3), 291–297 (1997)
32. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* **12**, 2825–2830 (2011)
33. Raghuathan, A., Steinhardt, J., Liang, P.: Certified defenses against adversarial examples. In: *Proceedings of the 6th International Conference on Learning Representations* (2018)
34. Ranzato, F., Zanella, M.: Abstract interpretation of decision tree ensemble classifiers. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. vol. 34, pp. 5478–5486 (2020)
35. Ranzato, F., Zanella, M.: Genetic adversarial training of decision trees. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. p. 358–367 (2021)
36. Roth, K., Kilcher, Y., Hofmann, T.: The odds are odd: A statistical test for detecting adversarial examples. In: *Proceedings of the 36th International Conference on Machine Learning*. pp. 5498–5507. PMLR (2019)
37. Schulze, J.P., Sperl, P., Böttinger, K.: DA3G: Detecting adversarial attacks by analysing gradients. In: *European Symposium on Research in Computer Security*. pp. 563–583 (2021)
38. Shi, T., Horvath, S.: Unsupervised learning with random forest predictors. *Journal of Computational and Graphical Statistics* (2006)
39. Sperl, P., Kao, C.Y., Chen, P., Lei, X., Böttinger, K.: DLA: dense-layer-analysis for adversarial example detection. In: *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. pp. 198–215 (2020)
40. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. In: *Proceedings of the 2nd International Conference on Learning Representations* (2014)
41. Tan, S., Soloviev, M., Hooker, G., Wells, M.T.: Tree space prototypes: Another look at making tree ensembles interpretable. In: *Proceedings of the 2020 ACM-IMS on Foundations of Data Science Conference*. pp. 23–34 (2020)
42. Tian, J., Zhou, J., Li, Y., Duan, J.: Detecting adversarial examples from sensitivity inconsistency of spatial-transform domain. In: *Proceedings of the AAAI Conference on Artificial Intelligence* (2022)
43. Vens, C., Costa, F.: Random forest based feature induction. In: *Proceedings of 11th IEEE International Conference on Data Mining*. pp. 744–753 (2011)
44. Vos, D., Verwer, S.: Efficient training of robust decision trees against adversarial examples. In: *Proceedings of the 38th International Conference on Machine Learning*. pp. 10586–10595 (2021)
45. Vos, D., Verwer, S.: Adversarially robust decision tree relabeling. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (2022)
46. Vos, D., Verwer, S.: Robust optimal classification trees against adversarial examples. In: *Proceedings of the AAAI Conference on Artificial Intelligence* (2022)
47. Wang, Y., Zhang, H., Chen, H., Boning, D., Hsieh, C.J.: On L_p -norm robustness of ensemble decision stumps and trees. In: *Proceedings of the 37th International Conference on Machine Learning*. pp. 10104–10114 (2020)

48. Yang, P., Chen, J., Hsieh, C.J., Wang, J.L., Jordan, M.: ML-LOO: Detecting adversarial examples with feature attribution. *Proceedings of the AAAI Conference on Artificial Intelligence* **34**(04), 6639–6647 (2020)
49. Yang, Y.Y., Rashtchian, C., Wang, Y., Chaudhuri, K.: Robustness for non-parametric classification: A generic attack and defense. In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research*, vol. 108, pp. 941–951 (2020)
50. Zhang, C., Zhang, H., Hsieh, C.J.: An efficient adversarial attack for tree ensembles. In: *Advances in Neural Information Processing Systems*. vol. 33, pp. 16165–16176 (2020)
51. Zhang, S., Chen, S., Liu, X., Hua, C., Wang, W., Chen, K., Zhang, J., Wang, J.: Detecting adversarial samples for deep learning models: A comparative study. *IEEE Transactions on Network Science and Engineering* **9**(1), 231–244 (2022)