Training Robust Tree Ensembles for Security

Yizheng Chen, Shiqi Wang, Weifan Jiang, Asaf Cidon and Suman Jana
Columbia University

***Abstract*—Tree ensemble models including random forests and gradient boosted decision trees, are widely used as security classifiers to detect malware, phishing, scam, social engineering, etc. However, the robustness of tree ensembles has not been thoroughly studied. Existing approaches mainly focus on adding more robust features and conducting feature ablation study, which do not provide robustness guarantee against strong adversaries.**

In this paper, we propose a new algorithm to train robust tree ensembles, by constructing tree structures that can integrate information from robustness of different features. Robust training can be formulated as a robust optimization process that maximizes the defender's gain when the adversary is trying to minimize that. Since the minimization problem is NP-hard, by analyzing the combinatorial structure of the problem, we design a general algorithm based on greedy heuristic to find better solutions to the problem than previous work. We implement the algorithm for gradient boosted decision trees in xgboost and random forests in scikit-learn. Our extensive evaluation over four benchmark datasets show that, we can train more robust models than the start-of-the-art robust training algorithm in gradient boosted decision trees, with a 1.26× increase in the minimal L_∞ evasion distance required for the strongest whitebox attacker. In addition, our algorithm is general across different gain metrics and performs equally well in random forest models. We achieve 3.32× increase in L_∞ robustness distance compared to the baseline random forest training method.

Furthermore, to make the robustness increase meaningful in security applications, we propose attack-cost-driven constraints for the robust training process. Our training algorithm maximizes attacker's evasion cost by integrating domain knowledge about feature manipulation costs. We use twitter spam detection as a case study to analyze attacker's cost increase to evade our robust model. We demonstrate that our technique can train robust model to rank robust features as most important ones, and our robust model requires about 8.4× increase in attacker's economic cost to be evaded compared to the baseline.

I. Introduction

Tree ensemble models such as random forest (RF) and gradient boosted decision trees (GBDT), have been widely used for different security tasks including detection of malware [37], phishing [24], [31], [21], and online fraud [39], [46], [35]. These models are widely used in production settings by many major corporations such as Uber [9], [11], Airbnb [10], Amazon [6] and Google Cloud [8] due to their high accuracy and simplicity. However, despite their popularity, the robustness of these models, unlike the neural networks, especially against a strong adversary is not very thoroughly studied [42], [33], [16]. Such robustness is especially important when these models are deployed in security-critical settings where adversaries try to evade them almost as soon as they are deployed.

Existing literature mainly use two different methods to increase and analyze the robustness of the tree ensemble models: selecting more robust features and feature category exclusion measurements. Several prior works argue that adding robust features increases the cost of model evasion, since intuitively the attacker needs to modify robust features that are more expensive to change than non-robust features [36], [39], [21], [46], [35], [32]. For example, it is more expensive for fraudsters to set up a reputable hosting server than registering a new cheap domain, and therefore models used for network security tasks that use Internet infrastructure features are relatively resilient to evasion. On the other hand, researchers have also argued that if a tree ensemble model's accuracy and false positive rate do not degrade significantly when certain feature categories are excluded, the model can be thought of as more robust against evasion attacks which modify the entire feature categories [37], [36], [35].

Unfortunately, such existing approaches are often not sufficient to systemically improve robustness for tree ensemble models. Defenders use both robust and non-robust features to achieve high accuracy and low false positive rate. If the trained model heavily relies on non-robust features to make accurate predictions, it can still be easily evaded despite the usage of robust features. On the other hand, the attacker can modify different categories of features altogether to evade the model, which is not captured by the threat model of feature exclusion study. We need methods to make the model internal structure robust given different types of features, with more emphasis on utilizing robust features than non-robust ones, and we need to quantify the minimal effort that the strongest attacker needs to make to evade the model.

In this paper, we propose a systematic method to train robust tree ensemble models for security, that integrates domain knowledge about feature robustness into constructing more robust model structure. We utilize recent work [16] that formulates a robust optimization procedure for training robust tree ensembles. Briefly, the regular training process of the decision trees in the ensemble aims to maximize the gain every time a node performs feature splitting (e.g., information gain, gini impurity reduction, loss reduction). In comparison, the robust training procedure maximizes the gain as if an arbitrary attacker is trying to minimize such gain at the same time, i.e., the inner minimization problem. As a new research area, there are many challenges in achieving strong robustness for tree ensembles. First, solving the inner minimization problem is NP-hard. Using ad-hoc heuristics is limited at achieving robustness across different gain metrics and different types of models [16]. In addition, the common robustness specification to increase the minimal L_∞ evasion distance does not capture attacker's evasion cost in security applications. The L_∞ distance assumes that every feature makes equal contribution to the evasion cost, which is often not the case for security classifiers. Moreover, it is unclear what the increased robustness distance means in terms of increase of the actual costs for an attacker.

To address the challenges, we first propose a general robust training algorithm based on greedy heuristic to provide better approximation to the inner minimization problem. We formulate the inner minimization as a 0-1 optimization problem, a special case of nonlinear integer program. It is an optimization problem that assigns integer values of 0 or 1 to variables in order to solve a minimization objective. By analyzing the problem structure, we design a robust training algorithm using greedy heuristic to approximate the solution to the NP-hard problem. Our algorithm constructs attacker's minimal gain by iteratively analyzing training data points when deciding each splitting node structure. Greedy heuristics are known to provide fast feasible solutions to classic optimization problems such as the knapsack problem [22], partition problem [30], set covering problem [20], with bounds on the approximation ratio to the optimal solution. Compared to previous work [16], our algorithm provides a better solution, and is general across different gain metrics and types of ensemble models.

Second, we propose a method to specify cost-driven robustness constraints in the robust training algorithm. We model attacker's cost in terms of changing different features, by categorizing features into low, medium, or high cost for attackers to increase or decrease the feature values. This can be done based on expert domain knowledge during feature selection stage. If a feature costs more to increase, we specify that the model must be robust against decreasing the feature value; or if a feature costs more to decrease, we specify the robustness against increasing the value. In addition, we give the attacker a larger perturbation budget for features that are easier to change, and smaller budget for more costly features. The cost-driven constraints help the model learn robustness that can maximize the cost of evasion for the attacker.

Lastly, we explain the robustness improvement of our model. Tree ensembles provide a unique type of model explainability that shows the feature importance ranking according to the model structure. We analyze whether the robust model ranks high cost features more important than low cost features, compared to the regularly trained model. Moreover, using the strongest whitebox attack against tree ensembles [33], we compare the minimal feature changes the attacker needs to make to evade the robust model and the baseline model. Based on the changes, we reason about the attacker's economic cost increase to evade the robust model.

We extensively evaluate the performance of our new robust training algorithm. We have implemented the algorithm in two types of tree ensembles with different gain metrics in the state-of-art tree ensemble learning libraries: gradient boosted decision trees that use arbitrary differentiable loss function in xgboost [18], and random forest that use gini impurity in scikit-learn [5]. In the gradient boosted decision trees evaluation, we compare against the regular training and state-of-the-art training algorithm from Chen et al. [16], over four benchmarking datasets including breast-cancer, cod-rna, ijcnn, and MNIST. Our robust training algorithm achieves on average 2.94× and 1.26× improvement over the baseline and state-of-the-art robust training algorithm respectively, in the minimal L_∞ distance required to evade the model. For the random forest model, we are the first to provide general robust training algorithm over arbitrary gain metrics, and therefore we evaluate against the regular training in scikit-learn. On average over the four benchmarking datasets, we achieve 3.32× robustness improvement in the minimal L_∞ evasion distance compared to the baseline.

To show how robust training can be useful for security applications, we use twitter spam detection as a case study. We reimplement the feature extraction over the dataset from [38] to detect malicious URLs posted by twitter spammers. We use four categories of features: shared resources-driven, heterogeneity-driven, flexibility related, and user account features. They capture that attackers reuse hosting infrastructure resources, use long redirection chains across different geographical locations, and prefer flexibility of deploying different URLs. We categorize the features into difficulty of changing the values based on domain knowledge. We use our cost-driven robustness specifications to train a robust model, and compare several aspects against the models trained from regular training. We obtain the following three observations. First, while the regularly trained model ranks low cost features to be the most important ones, the robust model ranks the most costly features as the most important ones. This shows that robust training can guide the model to utilize more information from robust features. Second, our robust model can increase the minimal sum of absolute feature changes (L_1 distance) required to evade the robust model by 5.5× compared to the baseline model. Lastly, we analyze the cost of such evasion and conclude that the attacker needs to spend 8.4× more money to evade the robust model while maintaining the same level of malicious activities. This shows promise to increase attacker's evasion cost for a wide range of security applications that use tree ensemble models.

Our contributions are summarized as the following:

- We design a new method to train robust tree ensembles for security applications. We propose a general robust training algorithm that can integrate attacker's cost-driven constraints. Our algorithm works across different gain metrics and tree ensemble types, and can maximize attacker's evasion cost.
- We implemented the robust training algorithm [1] for the gradient boosted decision trees in xgboost and the random forest in scikit-learn. Over four benchmarking datasets, our algorithm for xgboost is 2.94× and 1.26× more robust than the baseline and the state-of-the-art robust

[1] We are working on open sourcing our implementation.

algorithm respectively; and for scikit-learn it is 3.32× more robust than the baseline.

- Using twitter spam detection as a case study, we demonstrate how robust models can be trained to prefer features that are harder to perturb. In addition, we analyze attacker's evasion cost, and show that our robust model requires 8.4× more cost in dollar amount to be evaded compared to the baseline.

II. Background and Related Work

A. Tree Ensembles

A decision tree model uses logical predicates to provide predictions. Each internal node holds a predicate over some feature values. The tree structure guides the prediction path from the root to a leaf node containing the predicted value.

An ensemble of trees consists of multiple decision trees, which aggregates the predictions from individual trees. Popular aggregation functions include the average (random forest) and the sum (gradient boosted decision tree) of the prediction values from each decision tree. We use the following notations for the tree ensemble in this paper.

1) Notations: The training dataset D has N data points with d features $D = \{(x_i, y_i) | i = 1, 2, ..., N\} (x_i \in \mathbb{R}^d, y \in \mathbb{R})$. Each input x_i can be written as a d-dimensional vector, $x_i = [x_i^1, x_i^2, ..., x_i^d]$. A predicate p is in the form[2] of $x^j < \eta$, which evaluates the j-th feature x^j against the split threshold η. Specifically, for the i-th training data, the predicate checks whether $x_i^j < \eta$. If $p = true$, the decision tree guides the prediction path to the left child, otherwise to the right child. The prediction process repeats until x_i reaches a leaf node. We use a function f to denote a decision tree, which gives a real-valued output for the input data point x with the true label y. This captures not only regression trees, but also classification trees. For classification trees, $f(x)$ represents the predicted probability for the true label y.

The most common decision tree learning algorithms use a greedy strategy to recursively construct the nodes from the root to the leaves, e.g., notably CART [15], ID3 [44], and C4.5 [45]. The learning algorithm greedily picks the best feature j^* and the best split value η^* for each node, which partitions the data points that reach the current node (I) to the left child (I_L) and the right child (I_R), based on some score to maximize the *gain* of the split. In particular, under a split, $I = I_L \cup I_R$. Formally, the greedy algorithm optimizes the following objective.

$$j^*, \eta^* = \arg\max_{j,\eta} Gain(I_L, L_R) = \arg\max_{j,\eta} Gain(j, \eta, I) \tag{1}$$

$$Gain(j, \eta, I) = s(I) - s(I_L, I_R) \tag{2}$$

In Equation (2), s denotes a scoring function. For example, we can use Shannon entropy, Gini impurity, or any general loss function to evaluate the gain. After the split, the score becomes $s(I_L, I_R)$, which can be calculated as the weighted sum of the children scores. For example, using the Gini impurity, we have $Gain(j, \eta, I) = Gini(I) - Gini(I_L, I_R)$. A common strategy to solve Equation (1) is to enumerate all the features with all the possible split points to find the maximum gain. When the dataset becomes too large to compute efficiently, different optimization methods have been proposed to approximate Equation (1), e.g., weighted quantile sketch [18]. The learning algorithm chooses the best feature split with the maximum gain, and then recursively constructs the children nodes in the same way, until the score does not improve or some predetermined threshold (e.g., maximum depth) is reached.

A tree ensemble uses the weighted sum of prediction values from K decision trees, where K is usually a parameter specified by the user. Each decision tree can be represented as a function f_t. Then, the ensemble predicts the output $\hat{y}$ as follows.

$$\hat{y} = \phi(x) = a * \sum_{t=1}^{K} f_t(x) \tag{3}$$

Ensemble methods use bagging [12] or boosting [47], [25], [26] to grow the decision trees. The methods avoid overfitting and improves the performance of the tree ensemble over any single decision tree. Random forest and gradient boosted decision tree (GBDT) are the most widely used tree ensembles in the industry. The random forest model uses $a = \frac{1}{K}$, and the GBDT model set $a = 1$. They use different ensemble methods to grow trees in parallel or sequentially, which we describe next.

2) Random Forest: **Bagging.** A random forest model uses bagging [12] to grow the trees in parallel. Bagging, i.e., bootstrap aggregation, uses a random subset of the training data and a random subset of features to train individual learners.

For each decision tree f_t, we first randomly sample N' data points from D to obtain the training dataset $D_t = \{(x_i, y_i)\}$, where $|D_t| = N'$ and $N' \leq N$. The training data sample D_t for each decision tree can be drawn with or without replacement, depending on the implementation. In the case that $N' = N$, the data points are sampled with replacement. Then, at every step of the training algorithm that solves Equation (1), we randomly select d' features in I to find the optimal split, where $d' \leq d$. The feature sampling is repeated until we finish growing the decision tree. The training data and feature sampling are commonly called row sampling and column sampling, respectively, which help avoid overfitting of the tree ensemble.

The key distinguishing factors for a random forest model are the bagging method, and that decision trees are trained independently from each other. The gain in Equation (2) can be computed using any scoring function. We focus on the scikit-learn implementation of the random forest classifier since it is widely used by related work. In scikit-learn, a decision tree computes the predicted probability as the percentage of

[2] Oblique trees which use multiple feature values in a predicate is rarely used in an ensemble due to high construction costs [40].

training samples for each class reaching the leaf. It takes the class with the highest predicted probability to be the predicted class. The ensemble realizes the majority vote by taking the most likely class given averaged predicted values from the trees.

Random forest model has been used for various security applications, e.g., detecting malware distribution [37], malicious autonomous system [36], social engineering [39], phishing emails [24], [31], [21], advertising resources for ad blocker [32], and online scams [46], [35], etc. Researchers use feature importance ranking as a common approach to interpret the trained random forest model. In some cases, researchers have also analyzed the performance of the model (e.g., ROC curve) given different subsets of the features to reason about the predictive power of feature categories. We will describe the details of how the model-level feature importance is computed in Section II-A4.

3) Gradient Boosted Decision Tree: **Boosting.** Gradient boosted decision tree (GBDT) model uses boosting [47], [25], [26] to grow the trees sequentially. Boosting iteratively train the learners, improving the new learner's performance by typically reweighting the training data, so that data that is misclassified gets a higher weight. Gradient boosting generalizes the boosting method to use an arbitrarily differentiable loss function.

In this paper, we focus on the state-of-the-art GBDT training system xgboost [18], which has been used by data scientists daily in major companies such as Uber [9], [11], Airbnb [10], Amazon [6] and Google Cloud [8]. In the gradient boosting method, the decision trees are grown in an *additive* fashion. Specifically, when growing a new tree (f_t), all previous trees (f_1, f_2, ..., f_{t-1}) are fixed. Using $\hat{y}^{(t)}$ to denote the predicted value at the t-th iteration of adding trees, xgboost minimizes the regularized loss $\mathcal{L}^{(t)}$ for the entire ensemble, as the scoring function in Equation (2).

$$\begin{aligned}\mathcal{L}^{(t)} =& \sum_{i=1}^{n} l(y_i, \hat{y}^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) \\ =& \sum_{i=1}^{n} (l(y_i, \hat{y}^{(t-1)} + f_t(x_i))) + \Omega(f_t) + C\end{aligned} \tag{4}$$

In the equation, l is an arbitrary loss function, e.g., cross entropy; and $\Omega(f_i)$ is the regularization term, which captures the complexity of the i-th tree, and encourages simpler trees to avoid overfitting. The second line is derived by additive training, since $\hat{y}^{(t)} = \hat{y}^{(t-1)} + f_t(x_i)$ and $\sum_{i=1}^{t-1} \Omega(f_i)$ is a constant. In practice, using the Taylor series expansion for the loss function l at the point $\hat{y}^{(t-1)}$ to the second order can quickly and effectively optimize $\mathcal{L}^{(t)}$ (originally proposed in [27]). Thus, we have,

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n} (l(y_i, \hat{y}^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)) + \Omega(f_t) + C \tag{5}$$

Where $g_i = \partial_{\hat{y}^{(t-1)}}(l(y_i, \hat{y}^{(t-1)})$, and $h_i = \partial^2_{\hat{y}^{(t-1)}}(l(y_i, \hat{y}^{(t-1)})$. Note that in Equation (5), $\sum_{i=1}^{n} l(y_i, \hat{y}^{(t-1)})$ is a constant when growing the t-th tree f_t. Therefore, the training objective at the t-th iteration is to minimize:

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} (g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)) + \Omega(f_t) \tag{6}$$

Xgboost proposes the following regularization term, where T is the number of leaves in the t-th tree, w are the leaf weights used as the prediction, and γ and λ are hyperparameters.

$$\Omega(f_t) = \gamma T + \frac{1}{2} \lambda \|w\|^2 \tag{7}$$

The design of the regularization term by xgboost provides a closed form for the optimal leaf weights and minimal loss $\tilde{\mathcal{L}}^{(t)}_{struct}$ given the current tree structure, represented by T leaves where each leaf contains a set of training data points as I_j.

$$\tilde{\mathcal{L}}^{(t)}_{struct} = -\frac{1}{2} \sum_{j=1}^{T} \left[\frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} \right] + \gamma T \tag{8}$$

Using the optimal solution as the scoring function for Equation (2), the gain can be computed as:

$$\begin{aligned}&Gain(I_L, L_R) \\ &= \frac{1}{2} \left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma\end{aligned} \tag{9}$$

Boosting makes the newer tree dependent on previously grown trees. Each boosting round adds a new tree, and the user can specify number of rounds which determines the number of total decision trees in the ensemble. Previously, random forest was considered to generalize better than gradient boosting, since boosting alone could overfit the training data without tree pruning, whereas bagging avoids that. The regularization term introduced by xgboost effectively performs tree pruning during the training process, which improves the generalization performance of GBDT. In addition, xgboost supports bagging as well, including both row and column sampling.

4) Feature Importance: Among non-linear classifiers with strong predictive power, tree ensembles offer unique model-level feature importance ranking. Once trained, random forest and gradient boosted decision tree models have the same model structure. Therefore, feature importance ranking can be computed in the same way.

There are mainly two ways to measure the feature importance in a tree ensemble, the mean increase gain (MIG) and mean decrease accuracy (MDA). Mean increase gain is commonly named as mean decrease impurity, originally proposed in [15] since the decrease in gini impurity has been used as the increase in gain. MIG adds up the weighted average gain for a feature in every tree, where the weight is the percentage of samples that are split by the feature. The mean decrease accuracy (MDA) [13], [14] measures the average decrease in accuracy of the ensemble, when the values of a feature are randomly permuted in out-of-bag samples (e.g., validation set). We use mean increase gain (MIG) as the feature importance measure in this paper, as evaluated by related work.

B. Robust Tree Ensembles

We will first discuss two types of threat models under which we evaluate the robustness of tree ensembles, then describe the robust training framework that can achieve such robustness.

1) Threat Model: We evaluate the robustness of a tree ensemble by analyzing the potential evasion caused by the strongest possible adversary. We analyze the *minimal evasion distance* that the attacker needs to perturb in the features in order to evade the model.

Strongest whitebox attack. The strongest possible adversary against the tree ensembles is the Mixed Integer Linear Program (MILP) attacker [33]. The MILP attack assumes whitebox access to all the information about the ensemble model. The attack constructs a mixed integer linear program, where the variables of the program are nodes of the trees, the objective is to minimize a distance(e.g., L_∞) between the evasive example and the attacked data point, and the constraints of the program are based on the model structure. The constraints include model mislabel requirement, logical consistency among leaves and predicates. Using a solver, the MILP attack can find the minimal evasion distance. Otherwise, if the solver says the program is infeasible, there truly does not exist an adversarial example by perturbing the attacked data point. This can also be considered as the exact verification of the robust distance to a given data point that is classified correctly.

Attacker's budget. To evaluate the robustness of a model, we assume that the attacker can perturb every data point x_i given some budget, which bounds the adversarial data points at each node in a decision tree to a set $I' = \{(x'_i, y_i) | x'_i \in budget(x), \forall x \in I\}$.

L_∞ **ball.** The common way to specify the budget is to use a L_∞ norm-bounded ball. In related work [16], [17], $I' = \{(x'_i, y_i) | \eta - \epsilon \leq x'_i \leq \eta + \epsilon, \forall x \in I\}$

2) Robust Training: **Intuition.** The intuition for robust tree ensembles can be found in Figure 1. Here $x_1, ..., x_8$ are 8 training points belong to two different labels. For regular training in the upper side of Figure 1, we can easily find a split threshold η on feature x^j between x_4 and x_5 to perfectly separate all of the data points into left and right set with standard methods (e.g., information gain). However, when we consider the attacker's budget (e.g., bounded by L_∞ ball), x_4 and x_5 can be easily perturbed by the adversary and cross the splitting threshold η. Therefore, the accuracy is 100% while the worst case accuracy under adversary is 75% for split threshold η in regular training. Robust training will take attacker's budget into consideration and select the split threshold based on it. As a result, it can greatly improve the worst case accuracy under adversary but sacrificing certain accuracy as the cost. As shown in the lower side of Figure 1, the robust training will select the split threshold η' between x_3 and x_4 to maximize the worst case accuracy under adversary to 87.5%. As a tradeoff, x_4 will be wrongly separated and the regular accuracy becomes 87.5%. The advantage is that a most robust split can increase the minimal evasion distance (e.g., minimal L_∞) for the attacker.

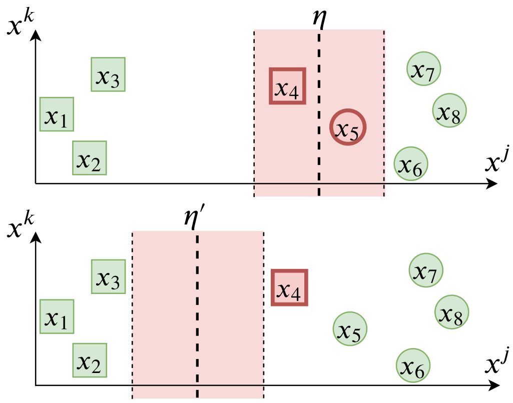


Fig. 1: A simple example showing the intuition behind the robust splitting during robust training. There are 8 training points with two different classes (square and circle). In the top figure, the split is 100% accurate but not robust given shaded region as attack budget. The accuracy under attack is reduced to 75%. The split in the bottom figure is always robust in the shaded region, but has a 87.5% accuracy.

Robust optimization. To achieve the robustness, the robust training algorithm needs to increase the accuracy of the model over the training data as much as possible, as if it is attacked under some allowable feature perturbations. Training a robust tree ensemble can be formulated as a robust optimization process, which increases the minimal evasion distance and the worst case accuracy under attack when generalizing the robustness to unseen test data.

Given the budget to perturb data points, the attacker reduces the maximum gain obtained in the regular training process (Equation (1)). Thus, instead of solving the maximization objective in Equation (1), we want to solve the following max-min problem, as formulated in [16].

$$j^*, \eta^* = \arg\max_{j,\eta} \min_{I'=\{(x'_i, y_i)\}} Gain(j, \eta, I') \qquad (10)$$

$$RS(j, \eta, I) = \min_{I'=\{(x'_i, y_i)\}} Gain(j, \eta, I') \qquad (11)$$

Specifically, $RS(j, \eta, I)$ denotes the solution to the inner minimization problem, namely the *robust score function*, which represents the gain of the split under potential attacks. Solving Equation (11) is NP-hard in general.

We propose cost-driven budget to specify the robustness, and a new general algorithm to solve Equation (11) in a greedy fashion, which provides better solutions. We allow the attacker to perturb every feature dimension with an arbitrary lower bound and upper bound, depending on the meaning of the feature. We will describe our algorithm to solve the inner minimization problem in Section III-B.

C. Related Work

Since the introduction of gradient tree boosting by Friedman [28], [29], gradient boosted decision tree (GBDT) training has been improved and adapted in many ways. PLANET [41] and pGBDT [49] parallelized the ensemble tree learning using methods including MapReduce and data partitioning. Ye et al. [51] proposed a stochastic gradient boosting algorithm for distributed decision trees. XGBoost [18] proposed a regularized objective for gradient tree boosting, and combined optimization on both the system level and algorithm level. LightGBM [34] used data sampling with large gradients and feature bundling with histogram-based algorithms to speed up the training time. Recently, researchers have proposed to adapt GBDT training for several stackable layers [23], high dimensional sparse output [48], and with training budget awareness [43], [50].

There are several attacks against ensemble trees. The strongest whitebox attack is the Mixed Integer Linear Program (MILP) attack [33]. The MILP attack formulates the evasion problem against ensemble trees as a mixed integer linear program. It uses predicates and leaves as the variables in the linear program to evade the ensemble model. The attack uses a solver to find the exact minimal changes to the features of a data point (e.g., L_∞ distance) to evade the model. If the solver says the program is infeasible to solve, there truly does not exist an adversarial example by perturbing the data point. Another whitebox attack proposed by Papernot et al. [42] is based on heuristics. The attack searches for leaves with different classes within the neighborhood of the targeted leaf of the benign example, to find a small perturbation that results in a wrong prediction. Among the blackbox attacks, Cheng et al.'s attack [19] has been demonstrated to work on ensemble trees. The attack minimizes the distance between a benign example and the decision boundary, using a zeroth order optimization algorithm with the randomized gradient-free method.

III. Methodology

In this section, we describe our methodology to train robust tree ensembles. As we have discussed in Section II-B, training robust tree ensemble needs to efficiently solve the inner minimization problem (Equation 11) in the robust optimization objective (Equation 10). We will first discuss how the inner minimization problem can be formulated as 0-1 integer optimization, then describe our practical algorithm based on greedy heuristic to find general solutions to the problem.

A. Optimization Formulation

The attacker's objective to minimize the node splitting gain (Equation (11)) can be formulated as a 0-1 integer optimization problem. Whenever we are trying to split along the j-th feature dimension, the training process first sorts all the data points that reach the current node along their x^j values. Figure 2 shows a simple example of nine data points numbered from 1 to 9, i.e. $I = \{x_1, x_2, ..., x_9\}$, with two classes shaped in circles and squares. The training process will try putting the splitting threshold η between every two consecutive data points, in order to get the splitting point with maximum gain (Equation (1)). In Figure 2, the split value under consideration is between data points x_5 and x_6. A regular training process then computes the gain of the split based on $I_L = \{x_1, x_2, x_3, x_4, x_5\}$ and $I_R = \{x_6, x_7, x_8, x_9\}$, using Equation 2. The robust training process considers the attacker's objective to minimize the gain as if some data points can be perturbed. If the attacker's budget is $[\eta - \epsilon_l, \eta + \epsilon_r]$ along the j-th feature, four data points could be perturbed to cross the splitting threshold η. We use ΔI to denote the set of uncertain points, $\Delta I = \{x_3, x_4, x_5, x_6\}$, which are the only points that can reduce the gain. Each point in ΔI can be assigned to either the left side I_L or the right side I_R, with $2^{|\Delta I|}$ possible assignments. Finding the minimal gain within all possible assignments is a combinatorial optimization problem. This becomes intractable as Equation (11) needs to be repeatedly solved during the training process.

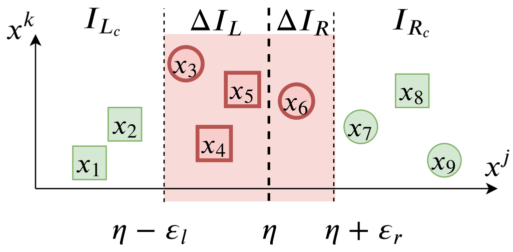


Fig. 2: A simple example to illustrate the uncertain set $\Delta I = \Delta I_L \cup \Delta I_R = [x_3, x_4, x_5, x_6]$ within the robust region $[\eta - \epsilon_l, \eta + \epsilon_r]$ on feature x^j. Splitting threshold η splits the data points into left set $I_L = \Delta I_L \cup I_{L_c}$ and right set $I_R = \Delta I_R \cup I_{R_c}$. The data points within the uncertain set ΔI can be perturbed to cross the splitting threshold by attacks while the data points outside the uncertain set $(I_{L_c} \cup I_{R_c})$ are certain to be robust under any possible attacks.

In general, this is a nonlinear 0-1 integer optimization problem. Let v_i denote whether an uncertain data point is assigned to the left side of the split, and let $(1 - v_i)$ represent whether the data point is assigned to the right side of the split. The inner minimization problem can be written as the following, given allowable attack budget defined in Section II-B1:

$$\begin{aligned} obj. &= \min_{I' = \{(x'_i, y_i)\}} Gain(j, \eta, I') \\ &= \min_{I' = \{(x'_i, y_i)\}} (s(I') - s(I'_L, I'_R)) \end{aligned} \tag{12}$$

Since $s(I')$ is a constant given j and η for the j-th feature, equivalently we have:

$$\begin{aligned} obj. &= \max_{I'_L, I'_R} s(I'_L, I'_R) \\ &= \max_{\Delta I_L, \Delta I_R} s(I_{L_c} \cup \Delta I_L, I_{R_c} \cup \Delta I_R) \\ &= \max_{v_i} s(I_{L_c} \cup \{v_i x_i | x_i \in \Delta I\}, \\ & \quad I_{R_c} \cup \{(1 - v_i) x_i | x_i \in \Delta I\}) \end{aligned} \tag{13}$$

We use I_{L_c} and I_{R_c} to represent points that are certainly assigned to the left side and right side, respectively. We use ΔI_L and ΔI_R to represent the uncertain points that are eventually put to the left and the right, and $\Delta I = \Delta I_L \cup \Delta I_R$. Using integer variables v_i, we can obtain the 0-1 optimization objective as shown in Equation (13). This optimization problem is NP-hard.

If we use any common function such as entropy, gini impurity, or cross entropy loss in Equation (13), it is immediate that this 0-1 optimization problem is nonlinear. To approximate the inner minimization problem, Chen et al. [16] use the minimal gain from four cases for GBDT: 1) put everything to the left, 2) put everything to the right, 3) swap ΔI_L and ΔI_R, and 4) the original split. For a single decision tree constructed using entropy and gini impurity, they use another heuristic to balance the label assignments as much as possible. Their method can increase the robustness of the ensemble compared to regular training, but does not scale across different types of split evaluation metrics and ensembles. Moreover, *simpler and better heuristics* exist that can provide better approximation for the 0-1 optimization problem. In the next section, we propose our general robust training algorithm that uses greedy heuristic to approximate the solution to the 0-1 optimization problem.

B. Robust Training Algorithm

We propose a general robust splitter algorithm based on greedy heuristic to find practical solutions to the 0-1 optimization objective (the inner minimization problem). Our algorithm works for different types of trees, including both classification and regression trees, different ensembles such as gradient boosted decision trees and random forest, and different splitting metrics used to compute the gain.

Problem structure. We recognize that this problem has a similar structure as the partition problem: partitioning a set containing positive integers into two subsets such that the sum of numbers from the two sets are equal. The problems are similar, but with several key differences. First, the two subsets already contain certain data points in I_{L_c} and I_{R_c}. Second, instead of summing up values from the subsets separately, we use a scoring function to evaluate the subsets altogether. Third, instead of optimizing for the sum values being equal, our objective is to maximize the output of the scoring function, or, equivalently minimizing the difference in the score of the entire set and the score of the two subsets. In combinatorial optimization, greedy heuristics often provide very good solutions to the problem in practice, with linear complexity instead of exponential by enumeration. Thus, we borrow the greedy heuristic to design a general robust splitter algorithm.

Algorithm 1 General Robust Splitter Algorithm

```
Input: training dataset $D = \{(x_i, y_i) | i = 1, 2, ..., N\}$
($x_i \in \mathbb{R}^d, y \in \mathbb{R}$).
Input: the set of data points reaching the current node
$I = \{(x_i, y_i)\}$, $|I| = m$.
Input: budget specification $\{(lower_j, upper_j) | j = 1, ..., d\}$.
Input: the score function $s$.
Output: the best split at the current node.
Initialize $RS^* = 0; j^* = 0; \eta^* = 0$
for $j = 1$ to $d$ do
    Sort $I = \{(x_i, y_i)\}$ along the j-th feature as $\{(x_{t_i}, y_{t_i})\}$

    for $t_i = t_1$ to $t_m$ do
        if $t_i = t_1$ then
            $\eta \leftarrow x^j_{t_1}$
        else
            $\eta \leftarrow \frac{1}{2}(x_{t_i} + x_{t_{i-1}})$
        end if
        $I_L = \{(x_i, y_i) | x^j_i < \eta - lower_j\}$
        $I_R = \{(x_i, y_i) | x^j_i > \eta + upper_j\}$
        $\Delta I = \{(x_i, y_i) | \eta - lower_j \leq x^j_i \leq \eta + upper_j\}$
        /* Greedily put $(x_k, y_k)$ to whichever side that has a
        smaller gain. */
        for every $(x_k, y_k)$ in $\Delta I$ do
            $LeftGain(j, \eta, I) = s(I) - s(I_L \cup \{(x_k, y_k)\}, I_R)$

            $RightGain(j, \eta, I) = s(I) - s(I_L, I_R \cup$
            $\{(x_k, y_k)\})$
            if $LeftGain < RightGain$ then
                $I_L = I_L \cup \{(x_k, y_k)\}$
            else
                $I_R = I_R \cup \{(x_k, y_k)\}$
            end if
        end for
        /* RS is the greedy minimal gain for the split $j, \eta$.
        */
        $RS(j, \eta, I) = s(I) - s(I_L, I_R)$
        /* Try to find the maximal of the minimal gain. */
        if $RS(j, \eta, I) > RS^*$ then
            $j^* = j; \eta^* = \eta$
        end if
    end for
end for
return $j^*$, $\eta^*$
```

Robust splitter algorithm. Algorithm 1 describes our general robust splitter algorithm. The algorithm provides the optimal splitting features j^* and the splitting threshold η^* as output. The input includes the training dataset, the set of data points that reach the current node $I = \{(x_i, y_i)\}$, the attacker's cost-driven budget specification for all the feature dimensions

$\{(lower_j, upper_j)|j = 1, ..., d\}$, and a score function s. The budget specification allows the j-th feature to be decreased by $lower_j$ and increased by $upper_j$ in the robust training process. Example score functions are the cross-entropy loss, Gini impurity, or Shannon entropy.

From `Line 10` to `Line 28`, the algorithm does robust training, and the loops outside that are the same procedure as used in the regular training. The algorithm marches through every feature dimension (the for loop at `Line 2`), to compute the minimal gain that can be caused by the attacker (`Line 24`) for every possible split on that feature dimension (the for loop at `Line 2`), and eventually returns the optimal split after the enumeration (`Line 31`). To enumerate the possible splits on the j-th feature dimension, we first sort all the data points along that dimension (`Line 3`). Then, we go through all the sorted data points (x_{t_i}, y_{t_i}) to consider the gain of a potential split $x^j < \eta$. If x_{t_i} is the smallest value on that dimension, $\eta = x_{t_i}$. Otherwise, η is the average of x_{t_i} and the previous value $x_{t_i - 1}$. Given attacker's budget on the j-th feature, we have three sets before computing the minimal gain: I_L contains the data points that stay on the left side of the split, I_R contains the data points that stay on the right, and ΔI contains all the uncertain points that could either be in the left or right if they are perturbed (`Line 10 to 12`). Next, from `Line 13` to `Line 22`, we go through every uncertain data point, and greedily put it to either I_L or I_R, whichever gives a smaller gain for the current split. After that, we compute the robust score $RS(j, \eta, I)$ at `Line 24` as the attacker's minimal gain, and update the optimal split j^*, η^* for the current node if such minimal gain is the largest (`Line 25 to 28`). Algorithm 1 solves the max min objective of Equation 10 when constructing the split for a node.

C. Attack Cost-driven Specification

One of the inputs to the Algorithm 1, the attacker's budget specification, is provided by the user of the robust training algorithm. We propose to specify attacker's budget driven by the cost of changing the features. In security applications, different features have different cost to be changed. Some features are more costly for attackers to perturb than others, e.g., it is more expensive to rent and set up a new server than simply getting a new phishing domain name. Increasing a feature value verses decreasing it may have very different costs for the attacker. For example, if a twitter spammer account sends more tweets, it becomes more suspicious and easier to be detected. Therefore, increasing the number of tweets costs more than decreasing it. Based on the security expert domain knowledge, we can translate the attacker's cost in perturbing the features into a high-dimensional box, which becomes the $lower_j$ and $upper_j$ for every feature in the training algorithm.

We use an interval for each feature to represent the lower bound and upper bound of allowable change. Specifically, for each feature dimension, we 1) identify the cost of changing the feature, 2) analyze the direction of the feature changes, and 3) use a larger bound for the easier change, and a smaller bound for the more costly change. We normalize all the feature values to be between 0 and 1, such that the interval bound for a feature represents a percentage of robust change that the training algorithm wants to achieve. By integrating attacker's cost into the training process, we can guide the model to be robust against changes within the bound. Therefore, to evade the robust model, attackers need to perturb the non-robust features more or switch to change the more robust features.

IV. Evaluation

In this section, we evaluate our robust tree ensemble training algorithm against the state-of-the-art robust and regular training methods. For the purpose of demonstration, we mainly focus on evaluating the Gradient Boosted Decision Trees (GBDT) and random forest models. However, our method is general across different types of tree ensembles and different gain metrics as we have discussed in Section III-B.

A. Benchmark Datasets

This section evaluates the robustness improvements in 4 benchmark datasets: breast cancer, cod-rna, ijcnn, and binary MNIST (2 vs. 6). The size of the training/testing sets and the number of features for these datasets are shown in Table III. For fair comparison, we follow the same experiment settings used in [16], including number of trees, maximum depth, and trained ϵ for L_∞-norm bound. We describe the details of each benchmark dataset below.

breast cancer. The breast cancer dataset [2] contains 2 classes of samples, each representing benign and malignant cells. The attributes represent different measurements of the cell's physical properties (e.g., the uniformity of cell size/shape).

cod-rna. The cod-rna dataset [3] contains 2 classes of samples representing sequenced genomes, categorized by the existence of non-coding RNAs. The attributes contain information on the genomes, including total free-energy change, sequence length, and nucleotide frequencies.

ijcnn1. The ijcnn1 dataset [4] is from the IJCNN 2001 Neural Network Competition. Each sample represents the state of a physical system at a specific point in a time series, and has a label indicating "normal firing" or "misfiring". The original 5 attributes in each sample are measurements of different properties of the physical system. Here, we use the 22-attribute version of ijcnn, a transformation of original data, since it turned out to achieve the best performance and won the competition.

MNIST 2 vs. 6. The binary mnist dataset [7] contains handwritten digits of "2" and "6". The attributes represent the gray levels on each pixel location.

Consistent with Chen et al.'s experiment settings [16], we randomly shuffle the test set, and generate advesarial examples for 100 test data points for breast cancer, ijcnn1, and binary MNIST, and 5000 test points for ijcnn1.

B. Implementation

We implement our robust training algorithm in xgboost [18] and scikit-learn [5]. The implementation in xgboost works with all their supported differentiable loss functions for both

Dataset	# of	Trained ϵ		Tree Depth			Test ACC (%)			Test FPR (%)			Avg. l_∞			Improv.	
	trees	Chen’s	ours	natural	Chen’s	ours	natural	Chen’s	ours	natural	Chen’s	ours	natural	Chen’s	ours	natural	Chen’s
breast-cancer	4	0.30	0.30	6	8	8	98.54	97.81	96.35	0.98	0.98	0.98	.2096	.3938	**.5146**	**2.58x**	**1.31x**
cod-rna	20	0.20	0.03	4	5	5	96.48	88.09	88.03	2.57	4.44	7.59	.0343	.0560	**.0692**	**2.02x**	**1.24x**
ijcnn1	60	0.20	0.02	8	8	8	97.91	96.03	93.65	1.64	2.15	1.62	.0268	.0326	**.0463**	**1.73x**	**1.42x**
MNIST 2 vs. 6	1,000	0.30	0.30	4	6	6	99.75	99.54	99.54	0.39	3.88	3.88	.0609	.3132	**.3308**	**5.43x**	**1.06x**

TABLE I: Test accuracy and robustness of GBDT models trained by our algorithm (ours), compared to regularly trained models (natural) and the models trained by Chen and Zhang et al.’s method [16] (Chen’s), in xgboost. We evaluate the model robustness with the average l_∞ distance of the adversarial examples found by Kantchelians MILP attack [33], the strongest whitebox attack. The improvement (Improv.) here denotes the average l_∞ robustness distance on our models over regularly trained ones and Chen and Zhang’s.

Dataset	# of	Trained ϵ	Tree Depth		Test ACC (%)		Test FPR (%)		Avg. l_∞		Improv.
	trees	ours	natural	ours	natural	ours	natural	ours	natural	ours	natural
breast-cancer	20	0.30	6	6	99.27	99.27	0.98	0.98	.2091	**.3783**	**1.81x**
cod-rna	60	0.20	14	14	96.55	89.11	2.97	5.28	.0335	**.0686**	**2.05x**
ijcnn1	80	0.10	14	14	97.90	92.24	1.54	0.06	.0278	**.1151**	**4.14x**
MNIST 2 vs. 6	60	0.30	14	14	99.55	99.30	0.38	0.48	.0484	**.2558**	**5.29x**

TABLE II: Test accuracy and robustness of random forest models trained by our algorithm (ours) compared to regularly trained models (natural), in scikit-learn. The improvement (Improv.) here denotes the average l_∞ robustness distance on our models over regularly trained ones.

Dataset	**Train set size**	**Test set size**	**# of features**
breast-cancer	546	137	10
cod-rna	59,535	271,617	8
ijcnn1	49,990	91,701	22
MNIST 2 vs. 6	11,876	1,990	784

TABLE III: Training and testing set sizes, and number of features for the four benchmark datasets.

the GBDT and random forest. For scikit-learn, we implement the robust training algorithm in random forest using the gini impurity score. In the following subsections, we evaluate the robustness of our GBDT and random forest models trained using xgboost and scikit-learn respectively.

C. GBDT Results

We first evaluate the robustness of our training algorithm on the gradient boosted decision trees (GBDT) models with the four aforementioned benchmark datasets. We measure the model robustness with the average L_∞ distance of the adversarial examples found by Kantchelian et al.’s MILP attack [33]. Note that Kantchelian’s MILP attack is the strongest adaptive attack under L_p-norm robustness distance for tree ensemble models. We compare the robustness of the model trained with our greedy algorithm against those trained from the regular training algorithm, as well as the state-of-the-art robust training algorithm proposed by Chen et al. [16].

As shown in Table I, the GBDT models trained with our greedy algorithms are more robust than the ones trained with regular training method and state-of-the-art robust training method. Specifically, it costs the Kantchelian et al.’s MILP attack 2.94× more L_∞ perturbation distance to evade our GBDT models than regularly trained ones. Compared to the state-of-the-art Chen and Zhang et al.’s robust training method [16], our models require on average 1.26× larger L_∞ perturbation distances while our models still maintain relatively high accuracy and low false positive rate. Note that the robustness improvement of our trained models are limited on binary MNIST dataset. This is because the trained and tested robustness ranges $L_\infty \leq 0.3$ are fairly large for MNIST dataset. The adversarial examples beyond that range are not imperceptible any more and thus the robustness becomes extremely hard to achieve without heavily sacrificing regular accuracy.

D. Random Forest Results

In this subsection, we evaluate the robustness of random forest models trained with our greedy algorithms on the four benchmark datasets. We implement our algorithm in a commonly used library scikit-learn [5]. to the best of our knowledge, no robust training methods are available under that setting using traditional gain metrics such as gini impurity and shannon entropy. Thus, we only compare against regular training algorithm for random forest models. We also measure the robustness of random forest models with the average L_∞ distance of adversarial examples found by Kantchelian et al.’s MILP attack [33].

As shown in Table II, the robustness of our random forest models significantly outperforms the regularly trained ones. Specifically, the average l_∞ distance of adversarial examples found by Kantchelian et al.’s MILP attack [33] is on average 3.32× larger than regular ones. On the other hand, there is only a 3.3% drop of test accuracy and a 0.2% increase of false positive rate on average for the robust models, although the underlying optimization is much harder for robust training than regular training.

E. Benefits of our greedy algorithms over existing heuristics

Here, we move one step further to provide insights on why our robust training algorithm outperforms the state-of-the-art Chen and Zhang et al.’s [16] and baseline training methods as shown in Section IV-C and IV-D. Essentially, the robustness improvement mainly benefits from our proposed greedy algorithm in solving the inner minimization problem of Equation (11).

According to Equation (11), our greedy algorithm is designed to minimize the gain for each feature split during the training process. The lower the gain is obtained by the algorithm, the stronger capability of the attacker is used for training, which guides the model to learn stronger robustness. Therefore, how well the algorithm can solve the minimization problem directly determines the eventual robustness the models can learn. To that end, we measure how our greedy algorithm performs in solving the gain minimization problem compared to the heuristics used in state-of-the-art Chen and Zhang's training algorithms [16] to illustrate its effectiveness.

Dataset	Better (%)	Equal (%)	Worse (%)	Total
breast-cancer	99.74	0.26	0	3,047
cod-rna	94.13	4.66	1.21	35,597
ijcnn	90.31	1.11	8.58	424,825
MNIST 2 vs. 6	87.98	6.33	5.69	796,264

TABLE IV: The percentage of the cases where our greedy algorithm performs better, equally well, or worse than the heuristics used in the state-of-the-art Chen and Zhang et al.'s robust training algorithms [16] in solving the maximization problem (Equation 2). The total number of cases represent the total number of splits evaluated during robust optimization.

On the four benchmark datasets, we measure the percentage of the cases where our greedy algorithms can better solve the maximization problem than the heuristics used in [16] and summarize the results in Table IV. The results show that our greedy algorithm can provide a better solution than heuristics used in Chen and Zhang et al.'s method [16] for at least 87.98% cases during the whole training process. On small datasets like breast-cancer and cod-rna, our algorithm performs equally or better for 100% and 98.79% cases respectively. Such significant improvements in solving the minimization problem greatly benefit the robustness of our trained models. The results provide insights on why our greedy-based robust training algorithm can obtain more robust tree ensembles than existing training methods.

V. Twitter Spam Case Study

In this section, we apply our robust tree ensemble training algorithm to a classic security application, spam URL detection on Twitter [38]. As a case study for security classifiers, we want to answer the following questions in the evaluation:

- **Cost-driven constraints:** How to specify the evasion cost-driven constraints to train robust tree ensembles?
- **Model explainability:** Does the robust model prefer more robust features compared to the baseline model?
- **Economic cost:** Against the strongest whitebox attack [33], does the robust model increase the economic cost for the attacker to evade it?

A. Dataset

We obtain the public dataset used in Kwon et al.'s work [38] to detect spam URLs posted on Twitter. Spammers spread harmful URLs on social networks such as Twitter to distribute malware, scam, or phishing content. These URLs go through a series of redirections, and eventually reach a landing page containing harmful content. The existing detectors proposed in prior works often make decisions based on content-based features that are strong in predictive power but easy to be changed, e.g., different words used in the spam tweet. Kwon et al. propose to use more robust features that incur monetary or management cost to be changed under adversarial settings. They extract these robust features from the URL redirection chains (RC) and the corresponding connected components (CC) formed by the chains.

Dataset	Training	Testing
Malicious	130,794	55,732
Benign	165,076	71,070
Total	295,870	126,802

TABLE V: The size of Twitter spam dataset [38].

Feature extraction. We reimplemented and extracted 25 features from the dataset in the original paper, as shown in Table VI. There are four families of features: shared resources-driven, heterogeneity-driven, flexibility-driven, and user account and post level features. The key intuitions behind the features are as follows. 1) Attackers reuse underlying hosting infrastructure to reduce the economic cost of renting and maintaining servers. 2) Attackers use machines hosted on bulletproof hosting services or compromised machines to operate the spam campaigns. These machines are located around the world, which tend to spread over larger geographical distances than benign hosting infrastructure, and it is hard for attackers to control the geographic location distribution of their infrastructure. 3) Attackers want to maximize the flexibility of the spam campaign, so they use many different initial URLs to make the posts look distinct, and different domains in the long redirection chains to be agile against takedowns. 4) Twitter spammers utilize specific characters to spread harmful content, such as hashtags and '@' mentions. We removed some highly correlated features from the original paper. For example, for a feature where the authors use both maximum and average numbers, we use the average number only.

Kwon et al. labeled the dataset by crawling suspended users, identifying benign users, and manually annotating tweets and URLs. In total, there are 186,526 distinct malicious tweets with spam URLs, and 236,146 benign ones. We randomly split the labeled dataset into 70% training set and 30% testing set as shown in Table V. We extract the aforementioned 25 features from each data point and normalize the values to be between 0 and 1 for training and testing.

B. Attack Cost-driven Constraints

In order to obtain the cost-driven constraints for robust training, we first analyze the cost of changing the features and the direction of the changes, then we specify robustness ranges accordingly.

1) Feature Analysis: We categorize the features into low, medium, and high cost to change. In total, sixteen features

Feature Name	**Description**	**Cost**	**Robustness Specification**
Shared Resources-driven Features			
EntryURLid	In degree of the largest redirector	Medium: Decreasing deg. increases # of redirector servers	[0.05, 0.2]
AvgURLid	Average in degree of URL nodes in the RC	Medium: Decreasing deg. increases # of redirector servers	[0.05, 0.2]
ChainWeight	Total frequency of edges in the RC	Low: Decreasing freq. decreases utility	[0.1, 0.2]
CCsize	# of nodes in the CC	Low: Decreasing number decreases flexibility	[0.1, 0.2]
CCdensity	Edge density of the CC	Low: Decreasing density decreases flexibility	[0.1, 0.2]
MinRCLen	Min length of the RCs in the CC	Low: Decreasing length decreases flexibility	[0.1, 0.2]
AvgLdURLDom	Average domain # of landing URL IPs in the CC	**High: Decreasing count increases BPH server cost**	[0, 0.2]
AvgURLDom	Average domain # for the IPs in the RC	Medium: Decreasing count increases server cost	[0.05, 0.2]
Heterogeneity-driven Features			
GeoDist	Total geo distance (km) traversed by the RC	**High: Changing dist. increases maintenance cost**	[0, 0.2]
CntContinent	# of unique continents in the RC	Medium: Decreasing count increases maintenance cost	[0.05, 0.2]
CntCountry	# of unique countries in the RC	Medium: Decreasing count increases maintenance cost	[0.05, 0.2]
CntIP	# of unique IPs in the RC	Low: Decreasing count decreases flexibility	[0.1, 0.2]
CntDomain	# of unique domains in the RC	Low: Decreasing count decreases flexibility	[0.1, 0.2]
CntTLD	# of unique TLDs in the RC	Low: Decreasing count decreases flexibility	[0.1, 0.2]
Flexibility Features			
ChainLen	Length of the RC	Low: Decreasing count decreases flexibility	[0.1, 0.2]
EntryURLDist	Distance from initial URL to the largest redirector	Low: Decreasing distance decreases flexibility	[0.1, 0.2]
CntInitURL	# of initial URLs in the CC	Low: Decreasing count decreases flexibility	[0.1, 0.2]
CntInitURLDom	Total domain name # in the initial URLs	Low: Decreasing count decreases flexibility	[0.1, 0.2]
CntLdURL	# of final landing URLs in the RC	Low: Decreasing count decreases flexibility	[0.1, 0.2]
AvgIPperURL	Average IP # per URL in the RC	Low: Decreasing count decreases flexibility	[0.1, 0.2]
AvgIPperLdURL	Average IP # per landing URL in the CC	**High: Increasing count increases BPH server cost**	[0.2, 0]
User Account Features			
Tweet Count	# of tweets made by the user account	Medium: Increasing count increases suspiciousness	[0.2, 0.05]
Mention Count	# of '@' count to mention other users	Low: Increasing count increases suspiciousness	[0.2, 0.1]
Hashtag Count	# of hashtags	Low: Increasing count increases suspiciousness	[0.2, 0.1]
URL Percent	Percentage of user posts that contain a URL	Low: Increasing count increases suspiciousness	[0.2, 0.1]

∗CC: connected component. RC: redirection chain. BPH: bulletproof hosting.

TABLE VI: We reimplement 25 features used in [38] to detect twitter spam. Among the four feature families, three features have high cost to change. To maintain the same level of spam activities, the attacker needs to purchase more bulletproof hosting servers to host the different landing pages if AvgLdURLDom feature is decreased or AvgIPperLdURL feature is increased. In addition, it is hard for the attacker to manipulate GeoDist feature.

have low cost, six have medium cost and only three features have high cost to change. We analyze the cost based on feature families as follows.

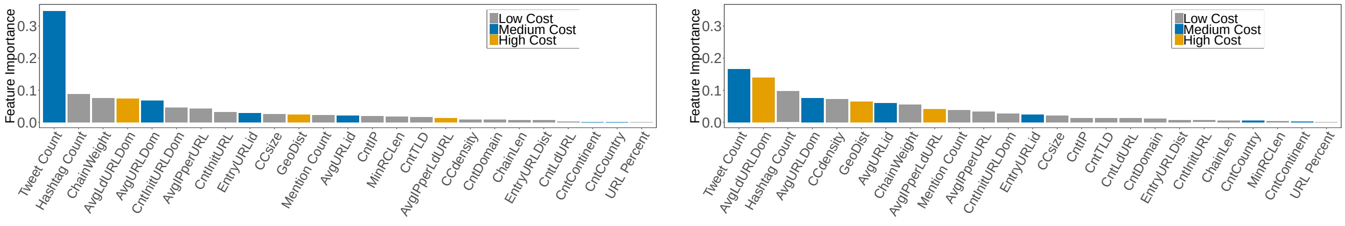
- **Shared resources:** All features cost more to be decreased than to be increased. If the attacker does not reuse the redirector in the chain as much as before, the attacker needs to set up additional redirector servers to maintain the same level of spam activities (EntryURLid and AvgURLid features). It costs even more to set up more servers for the landing pages, since the landing URLs contain actual malicious content, which are usually hosted on bulletproof hosting (BPH) services. Feature AvgLdURLDom captures how the attacker is reusing the malicious content hosting infrastructure. If the value is decreased, the attacker will need to set up more BPH severs, which has the highest cost in the category.
- **Heterogeneity:** The total geographical distance traversed by the URL nodes in the redirection chain has the highest cost to change in general (GeoDist). If the attacker uses all the available machines as resources for malicious activities, it is hard to control the location of the machines and the distance between them. Overall, it is harder to decrease GeoDist to what looks more like benign value than to increase it. Since GeoDist values for benign URL redirection chains are very concentrated in one or two countries, the attacker would need to purchase more expensive resources located close by to mimic benign URL. The other four features that count number of continents, countries, IPs, domains, and top-level domains incur cost for decreased flexibility and increased maintainence cost if the features are decreased.
- **Flexibility:** All features in this family except the last one have relatively low cost to decrease, because that decreases the flexibility of the attack. The high cost feature AvgIPperLdURL counts the number of IP addresses that host the malicious landing page URL. If the attacker wants more flexibility of hosting the landing page on more BPH servers, the cost will be increased significantly.
- **User account:** Increasing features in this family generally increases suspiciousness of the user account. Among them, increasing the tweet count is the most suspicious of all, since a tweet is capped by 140 characters which limits the number of mentions and hashtags, and percentage of posts containing URLs is also capped. If a user account sends too many tweets that puts the account to the top suspicious percentile, it can be easily detected by simple filtering mechanism and compromise the account.

Overall, three features have the highest cost to be perturbed: AvgLdURLDom, GeoDist[3], and AvgIPperLdURL. Decreasing AvgLdURLDom and increasing AvgIPperLdURL incurs cost

[3] GeoDist, CntContinent and CntCountry have similar intuition, but we choose GeoDist since it has finer granularity in feature values.

Dataset	# of trees		Trained ϵ	Tree Depth		Test ACC (%)		Test FPR (%)		Avg. l_1		Improv.
	natural	ours	ours	natural	ours	natural	ours	natural	ours	natural	ours	natural
Twitter Spam	30	150	various	8	24	99.38	95.59	0.89	5.07	.0069	**.0379**	**5.50x**

TABLE VII: Robustness improvement of twitter spam classification models. We use our new training algorithm to train a robust gradient boosted decision trees model, and compare it against the baseline model (natural). Results show that we achieve 5.5× robustness increase in the L_1 evasion distance against the strongest whitebox attacker [33].



(a) Baseline Model.

(b) Robust Model.

Fig. 3: The robust model ranks medium cost and high cost features higher than the baseline model. All three high cost features are ranked within top ten most important features for the robust model. In addition, the robust model utilizes top ten features with similar importance scores while the baseline model heavily relies on the top one features.

to obtain more bulletproof hosting servers for the landing page URL, and manipulating GeoDist is generally outside the control of the attacker. Other types of actions can also achieve the changes in AvgLdURLDom and AvgIPperLdURL, but it will generally decrease the profit of the malicious operation. To decrease AvgLdURLDom, if the attacker does not rent more BPH servers but only reduces the number of malicious landing pages, that reduces the profit. If the attacker increases the AvgIPperLdURL by using cheap servers, their malicious content could be taken down more often that interrupts the malicious operation.

2) Robustness Specification: We want to allow more perturbations for low cost features than high cost ones, and more perturbations on the low cost side (increase or decrease) than the more costly side. So we give larger ranges for low cost features, with direction of the ranges. Based on Table VI, we follow several guidelines to specify robustness bounds:

- If a feature costs more to be decreased, we specify a 0.2 robustness increase range. This guides the robust training process to obtain models that are robust against 20% increase in the feature value change, since it doesn't cost much for the attacker to do that. Similarly, if a feature costs more to be increased, we specify a 0.2 robustness decrease range.
- If a feature incurs cost to decrease, we give 0, 0.05, 0.1 robustness range to decrease that for low, medium, and high cost features, respectively. Similarly, we specify the features that have cost to increase.

C. Robust Model

Using the cost-driven robustness constraints, we trained a robust gradient boosted decision trees using our implementation in xgboost. We compare against the baseline model trained by the regular GBDT training method. Table VII summarizes the models.

Model performance. To evaluate the robustness, we run the strongest whitebox MILP attack that minimizes the L_1 distance of feature changes [33]. We run the attack from 100 randomly selected test data points. By perturbing the malicious points, the attack finds the exact minimal L_1 changes to generate adversarial examples for these data. On average, our robust model obtains 5.5× increase in the minimial L_1 evasion distance compared to the baseline. In the mean time, we keep reasonable performance for the robust model by increasing the model capacity. We achieve 95.59% accuracy and 5.07% false positive rate by increasing the number of trees and maximum tree depth used in the robust training process. It is a known tradeoff that increasing the robustness of a classifier may decrease its regular test accuracy and increase the false positive rate. In security applications, it is especially important to maintain low false positive rate to reduce the number of false alarms. Improving the false positive rate of the robust model will be a direction for future work.

Feature importance ranking. We compare the feature importance ranking between our robust model and the baseline to explain the robustness increase. Figure 3 shows that the model trained using our robust training algorithm ranks high cost features as more important ones, compared to the baseline model. The feature importance score is computed by the mean increase gain (MIG, described in Section II-A4), that aggregates the contribution of the loss reduction for splitting on the specific features across the ensemble. Figure 3a shows that the baseline model heavily relies on the Tweet Count features, such that it has a very high distinguishing feature importance score. Within top ten most important features for the baseline model, six of them are low cost features, and only one is high cost feature. On the other hand, Figure 3b shows that all three high cost features are within the top ten, and the number of low cost features are reduced to four for our robust model. The variance of MIG scores is smaller within the top ten features for the robust model compared to the baseline. This indicates that the robust model utilizes more information from various features and puts a heavier weight on the medium

Feature Name	Cost	Baseline	Our Robust Model
Shared Resources-driven Features			
EntryURLid	Medium: Decreasing	-0.05	0.0
AvgURLid	Medium: Decreasing	-0.02	1.25
ChainWeight	Low: Decreasing	1.44	17.97
CCsize	Low: Decreasing	-0.09	-0.14
CCdensity	Low: Decreasing	0.0	0.0
MinRCLen	Low: Decreasing	0.08	0.0
AvgLdURLDom	**High: Decreasing**	-0.17	**-111.41**
AvgURLDom	Medium: Decreasing	-0.3	-0.17
Heterogeneity-driven Features			
GeoDist	**High: Both Increasing and decreasing**	0.0	**504.17**
CntContinent	Medium: Decreasing	0.0	0.0
CntCountry	Medium: Decreasing	0.0	0.0
CntIP	Low: Decreasing	0.0	0.0
CntDomain	Low: Decreasing	0.0	0.0
CntTLD	Low: Decreasing	0.0	0.0
Flexibility Features			
ChainLen	Low: Decreasing	0.0	0.0
EntryURLDist	Low: Decreasing	0.0	0.0
CntInitURL	Low: Decreasing	0.44	0.01
CntInitURLDom	Low: Decreasing	0.0	0.0
CntLdURL	Low: Decreasing	0.0	0.0
AvgIPperURL	Low: Decreasing	0.0	0.0
AvgIPperLdURL	**High: Increasing**	0.0	**0.03**
User Account Features			
Mention Count	Low: Increasing	0.0	0.0
Hashtag Count	Low: Increasing	0.0	0.0
Tweet Count	Medium: Increasing	57.57	53.25
URL Percent	Low: Increasing	0.0	0.0

TABLE VIII: Comparison between average feature value changes in adversarial examples found by the strongest whitebox attack [33] against the baseline model and our robust model.

and high cost ones.

D. Cost Analysis

In the last section, we have shown that our new algorithm can train a robust model that ranks high cost features as the most importance ones, and increases the minimal L_1 evasion distances against the strongest whitebox attack, the MILP attack. Here, we analyze the meaning of increased robustness distance for attacker's economic cost.

Feature value changes. We record the average feature value changes for each feature dimension, to evade the baseline and the robust model against the MILP attack in Table VIII. We highlight the high cost features in bold in the table. For the baseline model, most features are not perturbed, and the majority of perturbed features have very small changes, from -0.17 to 1.44. Only the Tweet Count feature is increased by 57. This is consistent with the observation that the Tweet Count feature is ranked as the most important in Figure 3a for the baseline model. On the other hand, as shown in the column of our robust model in Table VIII, the large feature changes are concentrated on AvgLdURLDom, GeoDist, and the Tweet Count, where two of them are high cost features. On average, the attacker needs to at least reduce AvgLdURLDom by 111, which increases the number of bulletproof hosting servers needed to deploy these 111 domains. Then the attacker needs to increase GeoDist by 504, and increase the Tweet Count by 53 among several other feature changes.

Economic cost. Next, we analyze how much economic cost our robust model incurs for the attacker to evade it. We focus on translating the cost of AvgLdURLDom to the cost of BPH service, since it dominates the cost in the result. We assume that the attacker wants to maintain the same number of malicious landing pages to keep the same level of malicious activities. In order to evade the robust model, the attacker needs to reduce the AvgLdURLDom from the original of 126 to the 15 for adversarial examples. This can be achieved by increasing the BPH server IPs by 8.4×, calculated from $\frac{126}{15}$, such that the existing domains can be redistributed to the new servers. The average cost across 5 BPH service providers is \$314/month [1]. If the attacker wants to change this feature by using more BPH servers, this translates to \$2637/month cost, an increase by \$2323/month for each landing IP address that uses BPH hosting service. It is possible that the attacker wants to avoid spending this much money to rent more expensive BPH hosting servers, and thus will need to decrease the number of malicious landing pages by 8.4×. Spammers use different landing pages to attract more users and avoid detection. Reducing the number of landing pages can reduce the profit of the spam activity. If we assume a linear relationship between the profit and the number of landing pages, this means that the attacker's profit is reduced to $\frac{1}{8}$, which is another type of economic cost.

Other attacks. We conduct our analysis using the MILP attack that minimizes the L_1 norm. MILP attack is the strongest adaptive attack against tree ensembles based on L_p norm. The attacker can adapt to minimize the cost of evasion directly. However, this requires a more complicated modeling of the evasion cost, which needs to consider indirect cost not captured by the features, e.g., profit of different malicious activity levels, change of suspiciousness and chances of being detected, etc. We plan to explore these as future work.

VI. Conclusion

In this paper, we have designed, implemented, and evaluated a robust training method for tree ensembles that can integrate domain knowledge about feature evasion costs. Our robust training algorithm is based on greedy heuristic to approximate the solution to the NP-hard problem of optimizing attacker's evasion objective. We have evaluated over four benchmark datasets against the regular training method and the state-of-the-art robust training algorithm. Our results show that our model is 1.26× more robust than the state-of-the-art algorithm in gradient boosted decision trees, and 3.32× most robust than the regular random forest training algorithm, against the strongest whitebox attack based on L_p norm. Moreover, we use a case study to demonstrate that by specifying attack cost driven constraints in the training algorithm, we can increase attacker's economic cost of evading the robust model by 8.4×.

References

[1] Black-markt ecosystem. Estimating the cost of "Pwnership". https://www2.deloitte.com/content/dam/Deloitte/us/Documents/risk/us-risk-black-market-ecosystem.pdf.

[2] Breast Cancer Wisconsin (Original) Data Set. https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original).

[3] Cod-RNA Data Set. https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#cod-rna.

[4] Ijcnn1 Data Set. https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#ijcnn1.

[5] scikit-learn: Machine Learning in Python. https://scikit-learn.org/.

[6] Simplify machine learning with XGBoost and Amazon SageMaker. https://aws.amazon.com/blogs/machine-learning/simplify-machine-learning-with-xgboost-and-amazon-sagemaker/.

[7] The MNIST Database of Handwritten Digits. http://yann.lecun.com/exdb/mnist/.

[8] Training with XGBoost on AI Platform. https://cloud.google.com/ml-engine/docs/scikit/training-xgboost.

[9] Uber Open Source: Catching Up with Felix Cheung, Data Platform Engineering Manager. https://eng.uber.com/uber-open-source-felix-cheung-inteview/.

[10] Using Machine Learning to Predict Value of Homes On Airbnb. https://medium.com/airbnb-engineering/using-machine-learning-to-predict-value-of-homes-on-airbnb-9272d3d4739d.

[11] We run, we improve, we scale: The XGBoost story at Uber. https://conferences.oreilly.com/strata/strata-ny/public/schedule/detail/77040.

[12] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

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

[14] L. Breiman. Manual on setting up, using, and understanding random forests v3. 1. *Statistics Department University of California Berkeley, CA, USA*, 1:58, 2002.

[15] L. Breiman, J. Friedman, R. Olshen, and C. Stone. Classification and regression trees. *Wadsworth Int Group*, 37(15):237–251, 1984.

[16] H. Chen, H. Zhang, D. Boning, and C.-J. Hsieh. Robust decision trees against adversarial examples. In *International Conference on Machine Learning (ICML)*, 2019.

[17] H. Chen, H. Zhang, S. Si, Y. Li, D. Boing, and C.-J. Hsieh. Robustness verification of tree-based models. In *Advances in Neural Information Processing Systems*, 2019.

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

[19] M. Cheng, T. Le, P.-Y. Chen, J. Yi, H. Zhang, and C.-J. Hsieh. Query-efficient hard-label black-box attack: An optimization-based approach. In *International Conference on Learning Representations (ICLR)*, 2019.

[20] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.

[21] A. Cidon, L. Gavish, I. Bleier, N. Korshun, M. Schweighauser, and A. Tsitkin. High precision detection of business email compromise. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 1291–1307, 2019.

[22] G. B. Dantzig. Discrete-variable extremum problems. *Operations research*, 5(2):266–288, 1957.

[23] J. Feng, Y. Yu, and Z.-H. Zhou. Multi-layered gradient boosting decision trees. In *Advances in neural information processing systems*, pages 3551–3561, 2018.

[24] I. Fette, N. Sadeh, and A. Tomasic. Learning to detect phishing emails. In *Proceedings of the 16th international conference on World Wide Web*, pages 649–656. ACM, 2007.

[25] Y. Freund. Boosting a weak learning algorithm by majority. *Information and computation*, 121(2):256–285, 1995.

[26] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[27] J. Friedman, T. Hastie, R. Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

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

[29] J. H. Friedman. Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4):367–378, 2002.

[30] R. L. Graham. Bounds on multiprocessor timing anomalies. *SIAM J. Appl. Math.*, 17:263–269, 1969.

[31] G. Ho, A. Cidon, L. Gavish, M. Schweighauser, V. Paxson, S. Savage, G. M. Voelker, and D. Wagner. Detecting and characterizing lateral phishing at scale. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 1273–1290, 2019.

[32] U. Iqbal, P. Snyder, S. Zhu, B. Livshits, Z. Qian, and Z. Shafiq. Adgraph: A graph-based approach to ad and tracker blocking. In *Proc. of IEEE Symposium on Security and Privacy*, 2020.

[33] A. Kantchelian, J. Tygar, and A. Joseph. Evasion and hardening of tree ensemble classifiers. In *International Conference on Machine Learning*, pages 2387–2396, 2016.

[34] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3146–3154, 2017.

[35] A. Kharraz, W. Robertson, and E. Kirda. Surveylance: automatically detecting online survey scams. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 70–86. IEEE, 2018.

[36] M. Konte, R. Perdisci, and N. Feamster. Aswatch: An as reputation system to expose bulletproof hosting ases. *ACM SIGCOMM Computer Communication Review*, 45(4):625–638, 2015.

[37] B. J. Kwon, J. Mondal, J. Jang, L. Bilge, and T. Dumitraş. The dropper effect: Insights into malware distribution with downloader graph analytics. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1118–1129. ACM, 2015.

[38] H. Kwon, M. B. Baig, and L. Akoglu. A domain-agnostic approach to spam-url detection via redirects. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 220–232. Springer, 2017.

[39] T. Nelms, R. Perdisci, M. Antonakakis, and M. Ahamad. Towards measuring and mitigating social engineering software download attacks. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 773–789, 2016.

[40] M. Norouzi, M. Collins, M. A. Johnson, D. J. Fleet, and P. Kohli. Efficient non-greedy optimization of decision trees. In *Advances in neural information processing systems*, pages 1729–1737, 2015.

[41] B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo. Planet: massively parallel learning of tree ensembles with mapreduce. *Proceedings of the VLDB Endowment*, 2(2):1426–1437, 2009.

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

[43] S. Peter, F. Diego, F. A. Hamprecht, and B. Nadler. Cost efficient gradient boosting. In *Advances in Neural Information Processing Systems*, pages 1551–1561, 2017.

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

[45] J. R. Quinlan. C 4.5: Programs for machine learning. *The Morgan Kaufmann Series in Machine Learning*, 1993.

[46] M. Z. Rafique, T. Van Goethem, W. Joosen, C. Huygens, and N. Nikiforakis. It's free for a reason: Exploring the ecosystem of free live streaming services. In *Proceedings of the 23rd Network and Distributed System Security Symposium (NDSS 2016)*, pages 1–15. Internet Society, 2016.

[47] R. E. Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

[48] S. Si, H. Zhang, S. S. Keerthi, D. Mahajan, I. S. Dhillon, and C.-J. Hsieh. Gradient boosted decision trees for high dimensional sparse output. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3182–3190. JMLR. org, 2017.

[49] S. Tyree, K. Q. Weinberger, K. Agrawal, and J. Paykin. Parallel boosted regression trees for web search ranking. In *Proceedings of the 20th international conference on World wide web*, pages 387–396. ACM, 2011.

[50] Z. E. Xu, M. J. Kusner, K. Q. Weinberger, and A. X. Zheng. Gradient regularized budgeted boosting. *arXiv preprint arXiv:1901.04065*, 2019.

[51] J. Ye, J.-H. Chow, J. Chen, and Z. Zheng. Stochastic gradient boosted distributed decision trees. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 2061–2064. ACM, 2009.