

TREANT: Training Evasion-Aware Decision Trees

Stefano Calzavara*, Claudio Lucchese*, Gabriele Tolomei[†], Seyum Assefa Abebe* and Salvatore Orlando*

*Ca' Foscari University of Venice, Italy

Email: {name.surname}@unive.it

[†]University of Padua, Italy

Email: gtolomei@math.unipd.it

Abstract—Despite its success and popularity, machine learning is now recognized as vulnerable to *evasion attacks*, i.e., carefully crafted perturbations of test inputs designed to force prediction errors. In this paper we focus on evasion attacks against decision tree ensembles, which are among the most successful predictive models for dealing with non-perceptual problems. Even though they are powerful and interpretable, decision tree ensembles have received only limited attention by the security and machine learning communities so far, leading to a sub-optimal state of the art for adversarial learning techniques. We thus propose TREANT, a novel decision tree learning algorithm that, on the basis of a formal threat model, minimizes an evasion-aware loss function at each step of the tree construction. TREANT is based on two key technical ingredients: *robust splitting* and *attack invariance*, which jointly guarantee the soundness of the learning process. Experimental results on three publicly available datasets show that TREANT is able to generate decision tree ensembles that are at the same time accurate and nearly insensitive to evasion attacks, outperforming state-of-the-art adversarial learning techniques.

I. INTRODUCTION

Machine Learning (ML) is increasingly used in several applications and different contexts. When ML is leveraged to ensure system security, such as in spam filtering and intrusion detection, everybody acknowledges the need of training ML models resilient to adversarial manipulations [1], [2]. Yet the same applies to other critical application scenarios in which ML is now employed, where adversaries may cause severe system malfunctioning or faults. For example, consider an ML model which is used by a bank to grant loans to inquiring customers: a malicious user may try to fool the model into illicitly qualifying him for a loan. Unfortunately, traditional ML algorithms proved vulnerable to a wide range of attacks, and in particular to *evasion attacks*, i.e., carefully crafted perturbations of test inputs designed to force prediction errors [3], [4], [5], [6].

To date, research on evasion attacks has mostly focused on linear classifiers [7], [8] and, more recently, on deep neural networks [9], [10]. Whereas deep learning obtained remarkable and revolutionary results on many perceptual problems, such as those related to computer vision and natural language understanding, *decision trees ensembles* are nowadays one of the best methods for dealing with non-perceptual problems, and are one of the most commonly used techniques in Kaggle competitions [11]. Decision trees are also *interpretable* models [12], yielding predictions which are human-understandable in terms of syntactic checks over domain features, which is particularly appealing in the security setting. Unfortunately,

despite their success, decision tree ensembles have received only limited attention by the security and machine learning communities so far, leading to a sub-optimal state of the art for adversarial learning techniques (see Section II-C).

In this paper, we propose TREANT,¹ a novel learning algorithm designed to build decision trees which are resilient against evasion attacks at test time. Based on a formal threat model, TREANT optimizes an evasion-aware loss function at each step of the tree construction [13]. This is particularly challenging to enforce correctly, considered the greedy nature of traditional decision tree learning [14]. In particular, TREANT has to ensure that the local greedy choices performed upon tree construction are not short-sighted with respect to the capabilities of the attacker, who has the advantage of choosing the best attack strategy based on the fully built tree. TREANT is based on the combination of two key technical ingredients: a *robust splitting* strategy for decision tree nodes, which reliably takes into account at training time the attacker's capability of perturbing instances at test time, and an *attack invariance* property, which preserves the correctness of the greedy construction by generating and propagating constraints along the decision tree, so as to discard splitting choices which might be vulnerable to attacks.

We finally deploy our learning algorithm within a traditional random forest framework [15] and show its predictive power on real-world datasets. Notice that, although there have been various proposals that tried to improve robustness against evasion attacks by using ensemble methods [16], [17], [18], [19], it was shown that ensembles of weak models are not necessarily strong [20]. We avoid this shortcoming by employing TREANT to train an ensemble of decision trees which are individually resilient to evasion attempts.

A. Roadmap

To show how TREANT improves over the state of the art, we proceed as follows:

- 1) We first review decision trees and decision tree ensembles, presenting a thorough critique of existing adversarial learning techniques for such models (Section II).
- 2) We introduce our formal threat model, discussing an exhaustive white-box attack generation method, which allows for an accurate evaluation of the performance of

¹The name comes from the role playing game "Dungeons & Dragons", where it identifies giant tree-like creatures.

decision trees under attack and proves scalable enough for our experimental analysis (Section III).

- 3) We present TREANT, the first tree learning algorithm which greedily, yet soundly, minimizes an evasion-aware loss function upon tree construction (Section IV).
- 4) We experimentally show that TREANT outperforms existing adversarial learning techniques on three publicly available datasets (Section V).

Our analysis shows that TREANT is able to build decision tree ensembles that are at the same time accurate and nearly insensitive to evasion attacks. Compared to the state of the art, TREANT exhibits a ROC AUC improvement against the strongest attacker ranging from $\approx 10\%$ to $\approx 20\%$.

II. BACKGROUND AND RELATED WORK

A. Supervised Learning

Let $\mathcal{X} \subseteq \mathbb{R}^d$ be a d -dimensional vector space of real-valued features. An instance $\mathbf{x} \in \mathcal{X}$ is a d -dimensional feature vector (x_1, x_2, \dots, x_d) representing an object in the vector space.² Each instance $\mathbf{x} \in \mathcal{X}$ is assigned a label $y \in \mathcal{Y}$ by some unknown function $g : \mathcal{X} \mapsto \mathcal{Y}$, called the *target* function. Starting from a set of hypotheses \mathcal{H} , the goal of a *supervised learning* algorithm is to find the function $\hat{h} \in \mathcal{H}$ that best approximates the target g . This is practically achieved through empirical risk minimization [21]; given a sample of correctly labeled instances $\mathcal{D} = \{(\mathbf{x}_1, g(\mathbf{x}_1)), \dots, (\mathbf{x}_n, g(\mathbf{x}_n))\}$ known as the *training set*, the empirical risk is defined by a loss function $\mathcal{L} : \mathcal{H} \times (\mathcal{X} \times \mathcal{Y})^n \mapsto \mathbb{R}^+$ measuring the cost of erroneous predictions, i.e., the cost of predicting $\hat{h}(\mathbf{x}_i)$ instead of the true label $g(\mathbf{x}_i)$, for all $(\mathbf{x}_i, g(\mathbf{x}_i)) \in \mathcal{D}$. Supervised learning thus amounts to finding:

$$\hat{h} = \operatorname{argmin}_{h \in \mathcal{H}} \mathcal{L}(h, \mathcal{D}).$$

The loss \mathcal{L} is often obtained by aggregating an instance-level loss $\ell : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}^+$. Here, we define \mathcal{L} as the sum of the instance-level losses: $\mathcal{L}(h, \mathcal{D}) = \sum_{(\mathbf{x}, y) \in \mathcal{D}} \ell(h(\mathbf{x}), y)$.

B. Decision Trees and Decision Tree Ensembles

A powerful set of hypotheses \mathcal{H} is the set of the *decision trees* [22]. We focus on binary decision trees, whose internal nodes perform thresholding over feature values. Such trees can be inductively defined as follows: a decision tree t is either a leaf $\lambda(\hat{y})$ for some label $\hat{y} \in \mathcal{Y}$ or a non-leaf node $\sigma(f, v, t_l, t_r)$, where $f \in [1, d]$ identifies a feature, $v \in \mathbb{R}$ is the threshold for the feature f and t_l, t_r are decision trees. At test time, an instance \mathbf{x} traverses the tree t until it reaches a leaf $\lambda(\hat{y})$, which returns the *prediction* \hat{y} , denoted by $t(\mathbf{x}) = \hat{y}$. Specifically, for each traversed tree node $\sigma(f, v, t_l, t_r)$, \mathbf{x} falls into the left tree t_l if $x_f \leq v$, and into the right tree t_r otherwise. We just write λ or σ to refer to some leaf or node of the decision tree when its actual content is irrelevant. The problem of learning an optimal decision tree is known to be

²For simplicity, we only consider numerical features over \mathbb{R} . However, our framework can be readily generalized to other use cases, e.g., categorical or ordinal features, which we support in our implementation and experiments.

Algorithm 1 BUILDTREE

```

1: Input: training data  $\mathcal{D}$ 
2:  $\hat{y} \leftarrow \operatorname{argmin}_y \mathcal{L}(\lambda(y), \mathcal{D})$ 
3:  $\sigma(f, v, \lambda(\hat{y}_l), \lambda(\hat{y}_r)), \mathcal{D}_l, \mathcal{D}_r \leftarrow \text{BESTSPLIT}(\mathcal{D})$ 
4: if  $\mathcal{L}(\sigma(f, v, \lambda(\hat{y}_l), \lambda(\hat{y}_r)), \mathcal{D}) < \mathcal{L}(\lambda(\hat{y}), \mathcal{D})$  then
5:    $t_l \leftarrow \text{BUILDTREE}(\mathcal{D}_l)$ 
6:    $t_r \leftarrow \text{BUILDTREE}(\mathcal{D}_r)$ 
7:   return  $\sigma(f, v, t_l, t_r)$ 
8: else
9:   return  $\lambda(\hat{y})$ 
10: end if

```

NP-complete [23], [24]; as such, a top-down greedy approach is usually adopted [14], as shown in Algorithm 1.

The function BUILDTREE takes as input a dataset \mathcal{D} and initially computes the label \hat{y} which minimizes the loss on \mathcal{D} for a decision tree composed of just a single leaf; for instance, when the loss is the Sum of Squared Errors (SSE), such label just amounts to the mean of the labels in \mathcal{D} . The function then checks if it is possible to grow the tree to further reduce the loss by calling a *splitting* function BESTSPLIT (Algorithm 2), which attempts to replace the leaf $\lambda(\hat{y})$ with a new sub-tree $\sigma(f, v, \lambda(\hat{y}_l), \lambda(\hat{y}_r))$. This sub-tree is greedily identified by choosing f and v from an exhaustive exploration of the search space consisting of all the possible features and thresholds, and with the predictions \hat{y}_l and \hat{y}_r chosen so as to minimize the global loss on \mathcal{D} . If it is possible to reduce the loss on \mathcal{D} by growing the new sub-tree, the tree construction is recursively performed over the subsets $\mathcal{D}_l = \{(\mathbf{x}, y) \in \mathcal{D} \mid x_f \leq v\}$ and $\mathcal{D}_r = \mathcal{D} \setminus \mathcal{D}_l$, otherwise the original leaf $\lambda(\hat{y})$ is returned. Real-world implementations of the algorithm typically use multiple stopping criteria to prevent overfitting, e.g., by bounding the tree depth, or by requiring a minimum number of instances in the datasets used in the recursive calls.

Random Forest (RF) and Gradient Boosting Decision Trees (GBDT) are popular ensemble learning methods for decision trees [15], [25]. RFs are obtained by independently training a set of trees \mathcal{T} , which are combined into the *ensemble predictor* \hat{h} , e.g., by using majority voting to assign the class label. Each $t_i \in \mathcal{T}$ is typically built by using bagging and per-node feature sampling over the training set. In GBDTs, instead, each tree approximates a gradient descent step along the direction of loss minimization. Both methods are very effective, where RF is able to train models with low variance, while GBDTs are models of high accuracy yet possibly prone to overfit.

C. Related Work

Adversarial learning, which investigates the safe adoption of ML in adversarial settings [1], is a research field that has been consistently increasing of importance in the last few years. In this paper we deal with *evasion attacks*, a research sub-field of adversarial learning, where deployed ML models are targeted by attackers who craft adversarial examples that resemble normal data instances, but force wrong predictions. Most of the work in this field regards classifiers, in particular binary ones.

Algorithm 2 BESTSPLIT

- 1: **Input:** training data \mathcal{D}
 - ▷ Build a set of candidate tree nodes \mathcal{N} via an exhaustive search over f and v
 - 2: $\mathcal{N} \leftarrow \{\sigma(f, v, \lambda(\hat{y}_l), \lambda(\hat{y}_r)) \mid f \in [1, d] \wedge \exists(\mathbf{x}, y) \in \mathcal{D} : x_f = v \wedge \hat{y}_l, \hat{y}_r = \operatorname{argmin}_{y_l, y_r} \mathcal{L}(\sigma(f, v, \lambda(y_l), \lambda(y_r)), \mathcal{D})\}$
 - ▷ Select the candidate node $\hat{t} \in \mathcal{N}$ which minimizes the loss \mathcal{L} on the training data \mathcal{D}
 - 3: $\hat{t} = \operatorname{argmin}_{t \in \mathcal{N}} \mathcal{L}(t, \mathcal{D}) = \sigma(f, v, \lambda(\hat{y}_l), \lambda(\hat{y}_r))$
 - ▷ Split the training data \mathcal{D} based on the best candidate node $\hat{t} = \sigma(f, v, \lambda(\hat{y}_l), \lambda(\hat{y}_r))$
 - 4: $\mathcal{D}_l \leftarrow \{(\mathbf{x}, y) \in \mathcal{D} \mid x_f \leq v\}$
 - 5: $\mathcal{D}_r \leftarrow \mathcal{D} \setminus \mathcal{D}_l$
 - 6: **return** $\hat{t}, \mathcal{D}_l, \mathcal{D}_r$
-

The attacker starts from a positive instance that is classified correctly by the deployed ML model and is interested in introducing minimal perturbations on the instance to modify the prediction from positive to negative, thus “evading” the classifier [26], [3], [27], [28], [29], [30], [31], [10].

To prevent these attacks, different techniques have been proposed for different models, including support vector machines [8], [32], deep neural networks [33], [10], [34], and decision tree ensembles [29], [35]. Unfortunately, the state of the art for decision tree ensembles is far from satisfactory.

The first adversarial learning technique for decision tree ensembles is due to Kantchelian *et al.* and is called *adversarial boosting* [29]. It is an empirical data augmentation technique, borrowing from the *adversarial training* approach [9], where a number of evading instances are included among the training data to make the learned model aware of the attacks and, thereby, possibly more resilient to them. Specifically, at each boosting round, the training set is extended by crafting a set of possible perturbations for each original instance and by picking the one with the smallest margin, i.e., the largest misprediction risk, for the model trained that far. Adding perturbed instances to the training set forces the learning algorithm to minimize the *average* error over both the original instances and the chosen sample of evading ones, but this does not provide clear performance guarantees under attack. This is both because evading instances exploited at training time might not be representative of test-time attacks, and because optimizing the average case might not defend against the *worst-case* attack. Indeed, the experiments in Section V show that the performance of ensembles trained via adversarial boosting can be severely downgraded by evasion attacks.

The second adversarial learning technique for decision tree ensembles was proposed in a very recent work by Chen *et al.*, who introduced the first tree learning algorithm embedding the attacker directly in the optimization problem solved upon tree construction [35]. The key idea of their approach, called *robust trees*, is to redefine the splitting strategy of the training examples at a tree node. They first identify the so called *unknown* instances of \mathcal{D} , which may fall in either in \mathcal{D}_l or in \mathcal{D}_r , depending on adversarial perturbations. The authors thus claim that the optimal tree construction strategy would need to account for an exponential number of attack configurations over these unknown instances. To tame such algorithmic complexity, they propose a sub-optimal heuristic

approach based on four “representative” attack cases. Though the key idea of this algorithm is certainly interesting and shares some similarities with our own proposal, it also suffers from significant shortcomings. First, representative attack cases are not such anymore when the attacker is aware of the defense mechanism, and they are not anyway sufficient to subsume the spectrum of possible attacks: our algorithm takes into account all the possible attack cases, while being efficient enough for practical adoption. Moreover, the approach in [35] does not implement safeguards against the incremental greedy nature of decision tree learning: there is no guarantee that, once the best splitting has been identified, the attacker cannot adapt his strategy to achieve better results on the full tree. Indeed, the experimental evaluation in Section V shows that it is very easy to evade the trained models, which turn out to be even more fragile than those trained through adversarial boosting [29].

III. THREAT MODEL

The possibility to craft adversarial examples was popularized by Szegedy *et al.* in the image classification domain: their seminal work showed that it is possible to introduce minimal perturbations into an image so as to modify the prediction of its class by a deep neural network [9]. These evasion attacks questioned the applicability of ML to several security/business critical domains where malicious users can intentionally fool an ML model deployed online.

A. Loss Under Attack and Adversarial Learning

At an abstract level, we can see the attacker A as a function mapping each instance to a set of possible perturbations, which might be able to evade the ML model. Depending on the specific application scenario, not every attack is plausible, e.g., A cannot force some perturbations or behaves surreptitiously to avoid detection. For instance, in the typical image classification scenario, A is usually assumed to introduce just slight modifications that are perceptually undetectable to humans. This simple similarity constraint between the original instance \mathbf{x} and its perturbed variant \mathbf{z} is well captured by a distance [10], i.e., we might have $A(\mathbf{x}) = \{\mathbf{z} \mid \|\mathbf{z} - \mathbf{x}\|_\infty \leq \epsilon\}$.

Similarly, assuming that the attacker can run independent attacks on every instance of a given dataset \mathcal{D} , we can define $A(\mathcal{D})$ as the set of the datasets \mathcal{D}' which can be obtained by replacing each $(\mathbf{x}, y) \in \mathcal{D}$ with any (\mathbf{z}, y) such that $\mathbf{z} \in A(\mathbf{x})$.

The easiness of crafting successful evasion attacks defines the *robustness* of a given ML model at test time. The goal of learning a robust model is therefore to minimize the harm an attacker may cause via perturbations. This learning goal was formalized as a min-max problem by Madry *et al.* [13]:

$$\hat{h} = \operatorname{argmin}_{h \in \mathcal{H}} \underbrace{\max_{\mathcal{D}' \in A(\mathcal{D})} \mathcal{L}(h, \mathcal{D}')}_{\mathcal{L}^A(h, \mathcal{D})}. \quad (1)$$

The inner maximization problem models the attacker A replacing all the given instances with an adversarial example aimed at maximizing the loss. We call *loss under attack*, noted $\mathcal{L}^A(h, \mathcal{D})$, the solution to the inner maximization problem. The outer minimization resorts to the empirical risk minimization principle, aiming to find the hypothesis that minimizes the loss under attack on the training set.

B. Attacker Model

Distance-based constraints for defining the attacker’s capabilities are very flexible for perceptual problems and proved amenable for heuristic algorithms for solving the inner maximization problem of Equation 1 [13]. However, they cannot be easily generalized to other realistic application scenarios, e.g., where perturbations are not symmetric, where the attacker may not be able to alter some of the features, or where categorical attributes are present. To overcome such limitations, we model the attacker A as a pair (R, K) , where R is a set of *rewriting rules*, defining how instances can be corrupted, and $K \in \mathbb{R}^+$ is a *budget*, limiting the amount of alteration the attacker can apply to individual instances. Each rule $r \in R$ has the form:

$$[a, b] \xrightarrow{f}_k [\delta_l, \delta_u],$$

where $[a, b]$ and $[\delta_l, \delta_u]$ are intervals on $\mathbb{R} \cup \{-\infty, +\infty\}$, with the former defining the *precondition* for the application of the rule and the latter defining the *magnitude* of the perturbation enabled by the rule; $f \in [1, d]$ is the index of the feature to corrupt; and $k \in \mathbb{R}^+$ is the *cost* of the rule. The semantics of the rewriting rule can be explained as follows: if an instance \mathbf{x} satisfies the condition $x_f \in [a, b]$, then the attacker can corrupt the instance \mathbf{x} by adding any $v \in [\delta_l, \delta_u]$ to x_f and spending k from the available budget. The attacker can corrupt each instance by using as many rewriting rules as desired in whatever order, up to budget exhaustion.

According to this attacker model, we define $A(\mathbf{x})$, the set of the attacks against an instance \mathbf{x} , as follows.

Definition 1 (Attacks). *Given an instance \mathbf{x} and an attacker $A = (R, K)$, we let $A(\mathbf{x})$ be the set of the attacks that can be obtained from \mathbf{x} , i.e., the set of the instances \mathbf{z} such that there exists a sequence of rewriting rules $r_1, \dots, r_n \in R$ and a sequence of instances $\mathbf{x}_0, \dots, \mathbf{x}_n$ where:*

- 1) $\mathbf{x}_0 = \mathbf{x}$ and $\mathbf{x}_n = \mathbf{z}$;
- 2) for all $i \in [1, n]$, the instance \mathbf{x}_{i-1} can be corrupted into the instance \mathbf{x}_i by using the rewriting rule r_i ;
- 3) the sum of the costs of r_1, \dots, r_n is not greater than K .

Notice that $\mathbf{x} \in A(\mathbf{x})$ for any attacker A by picking an empty sequence of rewriting rules.

We highlight that this rule-based attacker model includes novel attack capabilities like asymmetric perturbations, easily generalizes to categorical variables, and still covers or approximates standard distanced-based models. For instance, L^0 -norm attacker models where the attacker can corrupt at will a limited number of features can be easily represented [29]. The use of a budget is convenient to fine-tune the power of the attacker and enables the adoption of standard evaluation techniques for ML models under attack, like *security evaluation curves* [2].

C. Attack Generation

Computing the loss under attack \mathcal{L}^A is useful to evaluate the resilience of ML models to evasion attacks at test time; yet this might be intractable, since it assumes the ability to identify the most effective attack for all the test instances. This issue is thus typically dealt with by using a heuristic attack generation algorithm, e.g., the fast gradient sign method [10] or any of its variants, to craft adversarial examples which empirically work well. However, our focus on decision trees and the adoption of a rule-based attacker model enables an exhaustive attack generation strategy for the test set which, though computationally expensive, proves scalable enough for our experimental analysis and allows the actual identification of the most effective attacks. This enables the most accurate security assessment in terms of the actual value of \mathcal{L}^A .

We consider a *white-box* attacker model, where the attacker has a complete knowledge of the trained decision tree ensemble. We thus assume that the attacker exploits the knowledge of the structure of the trees in the targeted ensemble and, most importantly, of the features and thresholds which are actually used in the prediction process. Note that a decision tree ensemble induces a finite partitioning of the input vector space \mathcal{X} , defined by the features and thresholds used in the internal nodes of the trees in the ensemble, where instances falling in the same partition share the same prediction. This partitioning makes it possible to significantly reduce the set of attacks that are relevant to the computation of \mathcal{L}^A by considering at most one representative attacked instance for a given partition. We achieve this by a recursive algorithm that, for the sake of space, we just sketch below. For any given instance \mathbf{x} , we recursively apply all valid rules up to budget exhaustion. In doing so, the interval $[x_f + \delta_l, x_f + \delta_u]$ of each applied rewriting rule, is split into sub-intervals induced by the ensemble’s thresholds relative to feature f , and we generate a single attack for each of the sub-intervals, including the extremes $x_f + \delta_l, x_f + \delta_u$. Note that we include the extremes of the preconditions of the rewriting rules in the partitioning, as to make sure that all recursively applicable rules are considered. The above enumeration strategy makes sure that all relevant attacks, i.e, causing at least one internal node of the ensemble to invert its outcome, are generated.

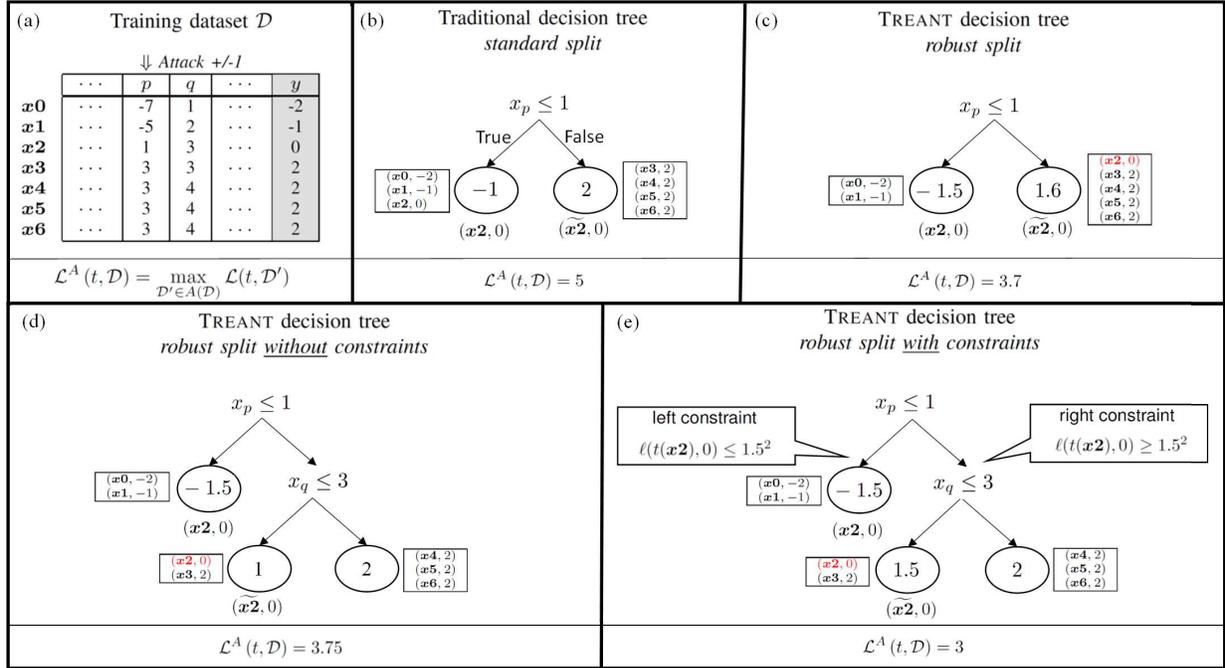


Fig. 1. Overview of the TREANT construction and its key challenges.

IV. TREANT: LEARNING ROBUST DECISION TREES

In this section, we present a novel decision tree learning algorithm that, by minimizing the loss under attack \mathcal{L}^A at training time, enforces resilience to evasion attacks at test time. We call TREANT the proposed algorithm.

A. Overview

Compared to Algorithm 1, TREANT replaces the BEST-SPLIT function by revising (i) the computation of predictions on the new leaves, (ii) the selection of the best split and (iii) the dataset partition along the recursion.

Before discussing the technical details, we build on the toy example in Figure 1 to illustrate the non-trivial issues arising when optimizing \mathcal{L}^A . Figure 1.(a) shows a dataset \mathcal{D} for which we assume the attacker $A = (\{r\}, 1)$, where r is a rewriting rule of cost 1 which allows the corruption of the feature p by adding any value in the interval $[-1, +1]$.

Assuming SSE is used as the underlying loss function \mathcal{L} , the decision stump initially generated by Algorithm 1 is shown in Figure 1.(b) along with the result of the splitting. Note that while the loss $\mathcal{L} = 2$ is small,³ the loss under attack $\mathcal{L}^A = 5$ is much larger.⁴ This is because the attacker may alter x_2

³ $\mathcal{L}(t, \mathcal{D}) = (-2 + 1)^2 + (-1 + 1)^2 + (-1 - 0)^2 + 4 \cdot (2 - 2)^2 = 2$.

⁴ $\mathcal{L}^A(t, \mathcal{D}) = (-2 + 1)^2 + (-1 + 1)^2 + (2 - 0)^2 + 4 \cdot (2 - 2)^2 = 5$.

into a perturbed instance \tilde{x}_2 so as to reverse the outcome of the test $x_p \leq 1$, i.e., the original instance x_2 falls into the left leaf of the stump, but the perturbed instance \tilde{x}_2 falls into the right leaf. The first issue of Algorithm 1 is thus that the estimated loss \mathcal{L} on the training set, computed when building the decision stump, is smaller than the loss under attack \mathcal{L}^A we would like to minimize. We solve this issue by designing a novel *robust splitting* strategy to identify the best split of \mathcal{D} , which directly minimizes \mathcal{L}^A when computing the leaves predictions and leads to the generation of a tree that is more robust to attacks. In particular, the decision stump learnt by using our robust splitting strategy is shown in Figure 1.(c), where the leaves predictions have been found by assuming that x_2 actually falls into the right leaf (according to the best attack strategy). For this new decision stump, the best move for the attacker is still to corrupt x_2 , but the resulting $\mathcal{L}^A = 3.7$ is much smaller than that of the previous stump.⁵ The figure also shows the outcome of the robust splitting.

However, a second significant issue arises when the decision stump is recursively grown into a full decision tree. Suppose to further split the right leaf of Figure 1.(c), therefore considering only the instances falling therein, including the instance x_2 put there by the robust splitting. We would find that the best split

⁵ $\mathcal{L}^A(t, \mathcal{D}) = (-2 + 1.5)^2 + (-1 + 1.5)^2 + (0 - 1.6)^2 + 4 \cdot (2 - 1.6)^2 = 3.7$.

is given by $x_q \leq 3$, where the feature q cannot be modified by the attacker. The resulting tree is shown in Figure 1.(d). Note however that, by creating the new sub-tree, new attacking opportunities show up, because the attacker now finds more convenient to just leave x_2 unaltered and let it fall directly into the left child of the root. As a consequence, by adding the new sub-tree, we observe an increased loss under attack $\mathcal{L}^A = 3.75$.⁶ This second issue can be solved by ensuring that any new sub-tree does not create new attacking opportunities that generate a larger loss. We call this property *attack invariance*. The proposed algorithm grows the sub-tree on the right leaf by carefully adjusting its predictions as shown in Figure 1.(e), still decreasing the loss under attack to $\mathcal{L}^A = 3$ with respect to the tree in Figure 1.(c).⁷ This is enforced by including constraints along the tree construction, as shown in the figure.

To sum up, the key technical ingredients of TREANT are:

- 1) *Robust splitting*: given a candidate feature f and threshold v , the robust splitting strategy evaluates the quality of the corresponding node split on the basis of a *ternary* partitioning of the instances falling into the node. It identifies those instances for which the outcome of the node predicate $x_f \leq v$ depends on the attacker’s moves, and those that cannot be affected by the attacker, thus always traversing the left or the right branch of the new node. In particular, the \mathcal{L}^A minimization problem is reformulated on the basis of left, right and unknown instances, i.e., instances which might fall either left or right depending on the attacker. Finally, the recursion on the left and right child of the node is performed by separating the instances in a binary partition based on the effects of the most harmful attack (Section IV-B).
- 2) *Attack invariance*: a security property requiring that the addition of a new sub-tree does not allow the attacker to find better attack strategies that increase \mathcal{L}^A . Attack invariance is achieved by imposing an appropriate set of constraints upon node splitting. New constraints are generated for each of the attacked instances present in the split node and are propagated to the child nodes upon recursion (Section IV-C).

The pseudo-code of the algorithm is given in Section IV-D. To assist the reader, the notation used in the present section is summarized in Table I.

B. Robust Splitting

We present our novel *robust splitting* strategy that grows the current tree t by replacing a leaf λ with a new sub-tree so as to minimize the loss under attack \mathcal{L}^A . For the sake of clarity, we discuss it as if the splitting was employed on the root node of a new tree, i.e., to learn the decision stump that provides the best loss reduction on the full input dataset \mathcal{D} . The next subsection discusses the application of the proposed strategy during the recursive steps of the tree-growing process.

⁶ $\mathcal{L}^A(t, \mathcal{D}) = (-2 + 1.5)^2 + (-1 + 1.5)^2 + (0 - 1.5)^2 + (2 - 1)^2 + 3 \cdot (2 - 2)^2 = 3.75$.

⁷ $\mathcal{L}^A(t, \mathcal{D}) = (-2 + 1.5)^2 + (-1 + 1.5)^2 + (0 - 1.5)^2 + (2 - 1.5)^2 + 3 \cdot (2 - 2)^2 = 3$.

TABLE I
NOTATION SUMMARY

Symbol	Meaning
\mathcal{D}	Training dataset
\mathcal{D}^λ	Local projection of \mathcal{D} on the leaf λ
$A(\mathbf{x})$	Set of all the attacks A can generate from \mathbf{x}
$A(\mathcal{D})$	Set of all the attacks A can generate from \mathcal{D}
$\lambda(\hat{y})$	Leaf node with prediction \hat{y}
$\sigma(f, v, t_l, t_r)$	Node testing $x_f \leq v$, having sub-trees t_l, t_r
$\mathcal{D}_l(f, v, A)$	Left elems of ternary partitioning on (f, v)
$\mathcal{D}_r(f, v, A)$	Right elems of ternary partitioning on (f, v)
$\mathcal{D}_u(f, v, A)$	Unkn. elems of ternary partitioning on (f, v)
$\mathcal{D}_L(\hat{t}, A)$	Left elems of robust splitting on \hat{t}
$\mathcal{D}_R(\hat{t}, A)$	Right elems of robust splitting on \hat{t}
$\mathcal{C}_L(\hat{t}, A)$	Set of constraints for the left child of \hat{t}
$\mathcal{C}_R(\hat{t}, A)$	Set of constraints for the right child of \hat{t}

Aiming at greedily optimizing the *min-max* problem in Equation 1, we have to find the best decision stump $\hat{t} = \sigma(f, v, \lambda(\hat{y}_l), \lambda(\hat{y}_r))$ such that:

$$\begin{aligned} \hat{t} &= \operatorname{argmin}_t \mathcal{L}^A(t, \mathcal{D}) = \\ &= \operatorname{argmin}_t \max_{\mathcal{D}' \in A(\mathcal{D})} \mathcal{L}(t, \mathcal{D}') = \\ &= \operatorname{argmin}_t \sum_{(\mathbf{x}, y) \in \mathcal{D}} \max_{z \in A(\mathbf{x})} \ell(t(\mathbf{z}), y). \end{aligned}$$

Whereas the pair (f, v) in $\hat{t} = \sigma(f, v, \lambda(\hat{y}_l), \lambda(\hat{y}_r))$ can be determined via an exhaustive search, the predictions \hat{y}_l and \hat{y}_r must be found by minimizing the loss under attack \mathcal{L}^A . However, this is not trivial, because the loss incurred by an instance (\mathbf{x}, y) may depend on the attacks it is possibly subject to. Similarly to [35], we thus define a ternary partitioning of the training dataset as follows.

Definition 2 (Ternary Partitioning). *For a feature f , a threshold v and an attacker A , the ternary partitioning of the dataset $\mathcal{D} = \mathcal{D}_l(f, v, A) \cup \mathcal{D}_r(f, v, A) \cup \mathcal{D}_u(f, v, A)$ is defined by:*

$$\begin{aligned} \mathcal{D}_l(f, v, A) &= \{(\mathbf{x}, y) \in \mathcal{D} \mid \forall z \in A(\mathbf{x}) : z_f \leq v\} \\ \mathcal{D}_r(f, v, A) &= \{(\mathbf{x}, y) \in \mathcal{D} \mid \forall z \in A(\mathbf{x}) : z_f > v\} \\ \mathcal{D}_u(f, v, A) &= (\mathcal{D} \setminus \mathcal{D}_l(f, v, A)) \setminus \mathcal{D}_r(f, v, A). \end{aligned}$$

In words, $\mathcal{D}_l(f, v, A)$ includes those instances (\mathbf{x}, y) falling into the left branch regardless of the attack, hence the attacker has no gain in perturbing x_f . A symmetric reasoning applies to $\mathcal{D}_r(f, v, A)$, containing those instances which fall into the right branch for all the possible attacks. The instances that the attacker may actually want to target are those falling into $\mathcal{D}_u(f, v, A)$, thus aiming at the largest loss. By altering those instances, the attacker may force each $(\mathbf{x}, y) \in \mathcal{D}_u(f, v, A)$ to fall into the left branch with a loss of $\ell(\hat{y}_l, y)$, or into the right branch, with a loss of $\ell(\hat{y}_r, y)$.

Example 1 (Ternary Partitioning). *The test node $x_p \leq 1$ and the attacker considered in Figure 1.(c) determine the following ternary partitioning of \mathcal{D} :*

- $\mathcal{D}_l(p, 1, A) = \{(\mathbf{x0}, -2), (\mathbf{x1}, -1)\}$
- $\mathcal{D}_r(p, 1, A) = \{(\mathbf{x3}, 2), (\mathbf{x4}, 2), (\mathbf{x5}, 2), (\mathbf{x6}, 2)\}$
- $\mathcal{D}_u(p, 1, A) = \{(\mathbf{x2}, 0)\}$

In other words, the instance $\mathbf{x2}$ is the only instance for which the branch taken at test time is unknown, as it depends on the attacker A .

By construction, given (f, v) , the loss \mathcal{L}^A can be affected by the presence of the attacker A only for the instances in $\mathcal{D}_u(f, v, A)$, while for all the remaining instances it holds that $\mathcal{L}^A = \mathcal{L}$. Since the attacker may force each instance of $\mathcal{D}_u(f, v, A)$ to fall into either the left or the right branch, the authors of [35] acknowledge a combinatorial explosion in the computation of \mathcal{L}^A . Rather than evaluating all the possible configurations, they thus propose a heuristic approach evaluating four “representative” attack cases: *i*) no attack, *ii*) all the unknown instances are forced in the left child, *iii*) all the unknown instances are forced in the right child, and *iv*) all the unknown instances are swapped by the attacker, i.e., they are forced in the left/right child when they would normally fall in the right/left child. Then, the loss \mathcal{L} is evaluated for these four split configurations and the maximum is used to estimate \mathcal{L}^A , so as to find the best stump \hat{t} to grow. Note that \mathcal{L} is computed as in a standard decision tree learning algorithm. Unfortunately, this heuristic strategy does not offer soundness guarantees, because the above four configurations leave potentially harmful attacks out of sight and do not induce an upper-bound of \mathcal{L}^A .

To avoid this soundness issue, while keeping the tree construction tractable, we pursue a *numerical optimization* as follows. For a given (f, v) , we highlight that finding the best attack configuration and finding the best left/right leaves predictions \hat{y}_l, \hat{y}_r are two inter-dependent problems, yet the strategy adopted in [35] is to first evaluate a few different attack configurations, and then to find the leaves predictions. We instead solve these two problems simultaneously by using a formulation of the min-max problem that, fixed (f, v) , is expressed solely in terms of \hat{y}_l, \hat{y}_r :

$$(\hat{y}_l, \hat{y}_r) = \underset{y_l, y_r}{\operatorname{argmin}} \mathcal{L}^A(\sigma(f, v, \lambda(y_l), \lambda(y_r)), \mathcal{D}), \quad (2)$$

where \mathcal{L}^A is decomposed via the ternary partitioning as:

$$\begin{aligned} \mathcal{L}^A(\sigma(f, v, \lambda(y_l), \lambda(y_r)), \mathcal{D}) &= \\ &= \mathcal{L}(\lambda(y_l), \mathcal{D}_l(f, v, A)) + \mathcal{L}(\lambda(y_r), \mathcal{D}_r(f, v, A)) + \\ &+ \sum_{(\mathbf{x}, y) \in \mathcal{D}_u(f, v, A)} \max\{\ell(y_l, y), \ell(y_r, y)\}. \end{aligned}$$

Observe that if the instance-level loss ℓ is convex, then \mathcal{L}^A is also convex⁸ and it can be efficiently optimized numerically. Convexity is indeed a property enjoyed by most loss functions such as SSE (for regression) and Log-Loss (for classification). This allows one to overcome the exploration of the exponential number of attack configurations, still finding the optimal solution (up to numerical approximation).

Given the best predictions \hat{y}_l, \hat{y}_r , we can finally produce a binary split of \mathcal{D} (as in Algorithm 1). To do this, we split

those instances by applying the best adversarial moves, i.e., by assuming that every $(\mathbf{x}, y) \in \mathcal{D}_u(f, v, A)$ is pushed into the left or right child so as to generate the largest loss. If the two children induce the same loss, then we assume the instance is not attacked.

Definition 3 (Robust Splitting). Given a decision stump to be grown $\hat{t} = \sigma(f, v, \lambda(\hat{y}_l), \lambda(\hat{y}_r))$ and an attacker A , the robust split of the dataset $\mathcal{D} = \mathcal{D}_L(\hat{t}, A) \cup \mathcal{D}_R(\hat{t}, A)$ is defined by:

- $\mathcal{D}_L(\hat{t}, A)$ contains all the instances of $\mathcal{D}_l(f, v, A)$ and $\mathcal{D}_R(\hat{t}, A)$ contains all the instances of $\mathcal{D}_r(f, v, A)$;
- for each $(\mathbf{x}, y) \in \mathcal{D}_u(f, v, A)$, the following rules apply:
 - if $\ell(\hat{y}_l, y) > \ell(\hat{y}_r, y)$, then (\mathbf{x}, y) goes to $\mathcal{D}_L(\hat{t}, A)$;
 - if $\ell(\hat{y}_l, y) < \ell(\hat{y}_r, y)$, then (\mathbf{x}, y) goes to $\mathcal{D}_R(\hat{t}, A)$;
 - if $\ell(\hat{y}_l, y) = \ell(\hat{y}_r, y)$, then (\mathbf{x}, y) goes to $\mathcal{D}_L(\hat{t}, A)$ if $x_f \leq v$ and to $\mathcal{D}_R(\hat{t}, A)$ otherwise.

Example 2 (Robust Splitting). Once identified \hat{y}_l and \hat{y}_r for the decision stump $\hat{t} = (p, 1, \lambda(-1.5), \lambda(1.6))$ in Figure 1.(c), the datasets obtained for the leaves by robust splitting are:

- $\mathcal{D}_L(\hat{t}, A) = \{(\mathbf{x0}, -2), (\mathbf{x1}, -1)\}$
- $\mathcal{D}_R(\hat{t}, A) = \{(\mathbf{x2}, 0), (\mathbf{x3}, 2), (\mathbf{x4}, 2), (\mathbf{x5}, 2), (\mathbf{x6}, 2)\}$

Notice that, unlike a standard decision tree learning algorithm, the right partition contains the instance $\mathbf{x2}$ due to the presence of the attacker, even though such instance satisfies the root node test.

To summarize, the ternary partitioning allows \mathcal{L}^A to be optimized for a given (f, v) and dataset \mathcal{D} , so as to find the best tree-growing step by an exhaustive search over f and v . Once this is done, the robust splitting allows the dataset \mathcal{D} to be partitioned in order to feed the algorithm recursion on the left and right children of the newly created stump. Ultimately, the goal of robust splitting is to solve the min-max problem of Equation 1 for a single tree-growing step, so as to find the best stump to be added to the tree, and push the attacked instances into the partition induced by the most harmful attack.

C. Attack Invariance

The optimization strategy described in Section IV-B needs some additional refinement to provide a sound optimization of \mathcal{L}^A on the full dataset \mathcal{D} . When growing a new sub-tree at a leaf λ , we denote with \mathcal{D}^λ the *local projection* of the full dataset at λ , i.e., the subset of the instances in \mathcal{D} falling in λ along the tree construction by applying the robust splitting strategy. The key observation now is that the robust splitting operates by assuming that the attacker behaves *greedily*, i.e., by locally maximizing the generated loss, but as new nodes are added to the tree, new attack opportunities arise and different traversal paths towards different leaves may become more fruitful to the attacker. If this is the case, the robust splitting becomes unrepresentative of the possible attacker’s moves and any learning decision made on the basis of such splitting turns out to be unsound, i.e., with no guarantee of minimizing \mathcal{L}^A . Notice that this is a major design flaw of the algorithm proposed in [35], and experimental evidence shows how the attacker can easily craft adversarial examples (see Section V).

⁸The pointwise maximum and the sum of convex functions preserve convexity.

In the end, the computation of the best split for a given leaf λ cannot be done just based on the local projection \mathcal{D}^λ , unless additional guarantees are provided. We thus enforce a security property called *attack invariance*, which ensures that the tree construction steps preserve the correctness of the greedy assumptions made on the attacker’s behavior. Given a decision tree t and an instance $(\mathbf{x}, y) \in \mathcal{D}$, we let $\Lambda^A(t, (\mathbf{x}, y))$ stand for the set of leaves of t which are reachable by some attack $z \in A(\mathbf{x})$ that generates the largest loss among $A(\mathbf{x})$.

Attack invariance requires that the tree construction steps preserves Λ^A , in that the attacker has no advantage in changing the attack strategy which was optimal up to the previous step, thus recovering the soundness of the greedy construction. We define attack invariance during tree construction as follows.

Definition 4 (Attack Invariance). *Let t be a decision tree and let t' be the decision tree obtained by replacing a leaf λ of t with the new sub-tree $\sigma(f, v, \lambda_l, \lambda_r)$. We say that t' satisfies attack invariance for the dataset \mathcal{D} and the attacker A iff:*

$$\forall (\mathbf{x}, y) \in \mathcal{D}^\lambda : \Lambda^A(t', (\mathbf{x}, y)) \cap \{\lambda_l, \lambda_r\} \neq \emptyset.$$

The above definition states that, after growing a new sub-tree from λ , the set of the best options for the attacker must include the newly created leaves, so that the path originally leading to λ still represents the most effective attack strategy against the decision tree.

Example 3 (Attack Invariance). *Let t be the decision tree of Figure 1.(c). Figure 1.(d) shows an example where adding a new sub-tree to t leads to a decision tree t' which breaks the attack invariance property. Indeed, we have $\Lambda^A(t', (\mathbf{x2}, 0)) = \{\lambda(-1.5)\}$, which contains neither $\lambda(1)$, nor $\lambda(2)$. Notice that the best attack strategy has indeed changed with respect to t , as leaving $\mathbf{x2}$ unaltered now produces a larger loss (2.25) than the originally strongest attack (1.0).*

We enforce attack invariance by introducing a set of *constraints* into the optimization problem of Equation 2. Suppose that the new sub-tree $\sigma(f, v, \lambda(\hat{y}_l), \lambda(\hat{y}_r))$ replaces the leaf λ and that an instance $(\mathbf{x}, y) \in \mathcal{D}^\lambda$ is placed in the right child by robust splitting, because one of its corruptions traverses the threshold v and $\ell(\hat{y}_r, y) \geq \ell(\hat{y}_l, y)$. Then, attack invariance is granted if, whenever the leaves $\lambda(\hat{y}_l)$ and $\lambda(\hat{y}_r)$ are later replaced by sub-trees t_l and t_r , there exists an attack $z \in A(\mathbf{x})$ that falls into a leaf of t_r generating a loss larger than (or equal to) the loss of any other attack falling in t_l . We enforce such constraint during the recursive tree building process as follows. The requirement $\ell(\hat{y}_r, y) \geq \ell(\hat{y}_l, y)$ is transformed in the pair of constraints $\ell(t_r(\mathbf{x}), y) \geq \gamma$ and $\ell(t_l(\mathbf{x}), y) \leq \gamma$, where $\gamma = \min\{\ell(\hat{y}_r, y), \ell(\hat{y}_l, y)\}$. These two constraints are propagated respectively into the recursion on the right and left children. As long as any sub-tree t_r replacing $\lambda(\hat{y}_r)$ satisfies the constraint $\ell(t_r(\mathbf{x}), y) \geq \gamma$ and any sub-tree t_l replacing $\lambda(\hat{y}_l)$ satisfies the constraint $\ell(t_l(\mathbf{x}), y) \leq \gamma$, the attacker has no advantage in changing the original attack strategy, hence attack invariance is enforced.

To implement this mechanism, each leaf λ is extended with a set of constraints, which is initially empty for the root of the tree. When λ is then split upon tree growing, the constraints therein are included in the optimization problem of Equation 2 to determine the best predictions \hat{y}_l, \hat{y}_r for the new leaves. These constraints are then (partially) propagated to the new leaves and new constraints are generated for them based on the following definition, which formalizes the previous intuition.

Definition 5 (Constraints Propagation and Generation). *Let λ be a leaf to be replaced with sub-tree $\hat{t} = \sigma(f, v, \lambda(\hat{y}_l), \lambda(\hat{y}_r))$ and let \mathcal{C} be its set of constraints. The sets of constraints $\mathcal{C}_L(\hat{t}, A)$ and $\mathcal{C}_R(\hat{t}, A)$ for the two new leaves are defined by:⁹*

- if $\ell(t(\mathbf{x}), y) \leq \gamma \in \mathcal{C}$ and there exists $z \in A(\mathbf{x})$ such that $z_f \leq v$, then $\ell(t_l(\mathbf{x}), y) \leq \gamma$ is added to $\mathcal{C}_L(\hat{t}, A)$;
- if $\ell(t(\mathbf{x}), y) \leq \gamma \in \mathcal{C}$ and there exists $z \in A(\mathbf{x})$ such that $z_f > v$, then $\ell(t_r(\mathbf{x}), y) \leq \gamma$ is added to $\mathcal{C}_R(\hat{t}, A)$;
- if $(\mathbf{x}, y) \in \mathcal{D}_u^\lambda(f, v, A) \cap \mathcal{D}_L^\lambda(\hat{t}, A)$, then $\ell(t_l(\mathbf{x}), y) \geq \ell(\hat{y}_r, y)$ is added to $\mathcal{C}_L(\hat{t}, A)$ and $\ell(t_r(\mathbf{x}), y) \leq \ell(\hat{y}_r, y)$ is added to $\mathcal{C}_R(\hat{t}, A)$;
- if $(\mathbf{x}, y) \in \mathcal{D}_u^\lambda(f, v, A) \cap \mathcal{D}_R^\lambda(\hat{t}, A)$, then $\ell(t_l(\mathbf{x}), y) \leq \ell(\hat{y}_l, y)$ is added to $\mathcal{C}_L(\hat{t}, A)$ and $\ell(t_r(\mathbf{x}), y) \geq \ell(\hat{y}_l, y)$ is added to $\mathcal{C}_R(\hat{t}, A)$.

Example 4 (Enforcing Constraints). *The tree in Fig. 1.(e) is generated by enforcing a constraint on the loss of $\mathbf{x2}$. After splitting the root, the constraint $\ell(t_r(\mathbf{x2}), 0) \geq \ell(\hat{y}_l, 0)$ is generated for the right leaf of the tree in Fig. 1.(c), where $\ell(\hat{y}_l, 0) = (-1.5 - 0)^2 = 2.25$. The solution of the constrained optimization problem on the right child of the tree in Fig. 1.(c) finally grows two new leaves, generating the tree in Fig. 1.(e). The difference from the tree in Fig. 1.(d) is that the prediction on the left leaf of the right child of the root has been enforced to satisfy the required constraint. For this tree, the attacker has no gain in changing attack strategy over the previous step of the tree construction, shown in Figure 1.(c).*

More formally, while for the tree t in Fig. 1.(c) we have $\Lambda^A(t, (\mathbf{x2}, 0)) = \{\lambda(1.6)\}$, after growing t with suitable constraints we obtain the tree t' in Fig. 1.(e), where the leaf $\lambda(1.6)$ has been substituted with a decision stump with the two new leaves $\{\lambda(1.5), \lambda(2)\}$. This entails $\Lambda^A(t', (\mathbf{x2}, 0)) = \{\lambda(-1.5), \lambda(1.5)\}$, where $\Lambda^A(t', (\mathbf{x2}, 0)) \cap \{\lambda(1.5), \lambda(2)\} = \{\lambda(1.5)\} \neq \emptyset$, thus satisfying the attack invariance property of Definition 4.

Constraints grant attack invariance at the cost of reducing the space of the possible solutions for tree-growing. Nevertheless, in the experimental section we show that this property does not prevent the construction of robust decision trees that are also accurate in absence of attacks.

D. Tree Learning Algorithm

Our TREANT construction is summarized in Algorithm 3. The core of the logic is in the call of the TSPLIT function (line 3), which takes as input a dataset \mathcal{D} , an attacker A and a set of

⁹We use the symbol \leq to stand for either \leq or \geq when the distinction is unimportant.

Algorithm 3 TREANT

```
1: Input: training data  $\mathcal{D}$ , attacker  $A$ , constraints  $\mathcal{C}$ 
2:  $\hat{y} \leftarrow \operatorname{argmin}_y \mathcal{L}^A(\lambda(y), \mathcal{D})$  subject to  $\mathcal{C}$ 
3:  $\sigma(f, v, \lambda(\hat{y}_l), \lambda(\hat{y}_r)), \mathcal{D}_l, \mathcal{D}_r, \mathcal{C}_l, \mathcal{C}_r \leftarrow \text{TSPLIT}(\mathcal{D}, A, \mathcal{C})$ 
4: if  $\mathcal{L}^A(\sigma(f, v, \lambda(\hat{y}_l), \lambda(\hat{y}_r)), \mathcal{D}) < \mathcal{L}^A(\lambda(\hat{y}), \mathcal{D})$  then
5:    $t_l \leftarrow \text{TREANT}(\mathcal{D}_l, A, \mathcal{C}_l)$ 
6:    $t_r \leftarrow \text{TREANT}(\mathcal{D}_r, A, \mathcal{C}_r)$ 
7:   return  $\sigma(f, v, t_l, t_r)$ 
8: else
9:   return  $\lambda(\hat{y})$ 
10: end if
```

constraints \mathcal{C} initially empty, and implements the construction detailed along the present section. The construction terminates when it is not possible to further reduce \mathcal{L}^A (line 4).

Function TSPLIT is summarized in Algorithm 4. Specifically, the function returns the sub-tree minimizing the loss under attack \mathcal{L}^A on \mathcal{D} subject to the constraints \mathcal{C} , based on the ternary partitioning (lines 2-3). It then splits \mathcal{D} by means of the robust splitting strategy (lines 4-5) and returns new sets of constraints (lines 6-7), which are used to recursively build the left and right sub-trees. The optimization problem (line 2) is numerically solved via the `scipy` implementation of the SLSQP (Sequential Least Squares Programming) method, which allows the minimization of a function subject to inequality constraints, like the constraint set \mathcal{C} generated/propagated by TREANT during tree growing.

There is an important point worth discussing about the implementation of the algorithm. As careful readers may have noticed, the TSPLIT function splits each leaf λ by relying on the set of attacks $A(\mathbf{x})$ for all instances $(\mathbf{x}, y) \in \mathcal{D}^\lambda$. Though one could theoretically pre-compute all the possible attacks against the instances in \mathcal{D} , this implementation would be very inefficient both in time and space, given the potentially huge number of instances and attacks. Our implementation, instead, *incrementally* computes a *sufficient* subset of $A(\mathbf{x})$ along the tree construction.

First, each instance (\mathbf{x}, y) is enriched with a cost annotation k , denoted by $(\mathbf{x}, y)^k$, initially set to 0 on the root. Such annotation keeps track of the cost of the adversarial manipulations performed to push (\mathbf{x}, y) into λ during the tree construction. When splitting the leaf λ on (f, v) , the algorithm generates only the attacks against the feature f which enforce *maximal* perturbations of x_f , as such maximal perturbations maximize the chance of crossing the threshold v without incurring in any extra cost. Moreover, the attack generation assumes that k was already spent from the attacker’s budget to further reduce the number of possible attacks. When the instance $(\mathbf{x}, y)^k$ is pushed into the left or right partition of \mathcal{D}^λ by robust splitting, the label k is updated to $k + k'$, where k' is the *minimum cost* the attacker must spend to achieve the desired node outcome. The same idea is applied when propagating constraints, which are also associated with specific instances (\mathbf{x}, y) for which the computation of $A(\mathbf{x})$ is required.

Observe that this implementation assumes that only the cost

of adversarial manipulations is relevant, not their magnitude, which is still sound when none of the corrupted features is tested multiple times on the same path of the tree. We enforce such restriction during the tree construction, which further regularizes the growing of the tree. Since we are eventually interested in decision tree ensembles, this does not impact on the performance of whole trained models.

V. EXPERIMENTAL EVALUATION

A. Methodology

We compare the performance of classifiers trained by different learning algorithms: two standard approaches, i.e., Random Forest [15] (RF) and Gradient Boosting Decision Trees [25] (GBDT) as provided by the LightGBM¹⁰ framework; two state-of-the-art adversarial learning techniques, i.e., Adversarial Boosting [29] (AB) and Robust Trees [35] (RT); and a Random Forest of trees trained using the proposed TREANT algorithm (RF-TREANT).¹¹ Notice that the original implementation of AB exploited a heuristic algorithm to find good adversarial examples, which does not guarantee to find the most damaging attack. Our own implementation of AB, which is built on top of LightGBM, exploits the white-box attack generation method described in Section III-C to find the *best* adversarial examples. In this regard, our implementation is thus more effective than the original algorithm.

We perform our experimental evaluation on three publicly available datasets, using three standard validity measures: accuracy, macro F1 and ROC AUC. We compute all measures both in absence of attacks and under attack, using our white-box attack generation method. We used a 60-20-20 train-validation-test split through stratified sampling. Hyperparameter tuning on the validation data was conducted to set the number of trees (≤ 100), number of leaves ($\{8, 32, 256\}$) and learning rate ($\{0.01, 0.05, 0.1\}$) of the various ensembles so as to maximize ROC AUC. All the results reported below were measured on the test data. Observe that all the compared adversarial learning techniques are parametric with respect to the budget granted to the attacker, modeling his power: we consider multiple instances of such budget both for training (*train budget*) and for testing (*test budget*).

The goal of our experimental evaluation is answering three key questions:

- 1) What is the performance of standard learning approaches like RF and GBDT when they are adopted in an adversarial setting?
- 2) What is the performance improvement achieved by the adoption of adversarial learning techniques for different test budgets?
- 3) What is the importance of the training budget on the performance of adversarial learning techniques?

¹⁰<https://github.com/microsoft/LightGBM>

¹¹The source code of TREANT is available at <https://github.com/omitted-for-anonymous-review>

Algorithm 4 TSPLIT

1: **Input:** training data \mathcal{D} , attacker A , constraints \mathcal{C} ▷ Build a set of candidate tree nodes \mathcal{N} using the ternary partitioning to optimize \mathcal{L}^A 2: $\mathcal{N} \leftarrow \{ \sigma(f, v, \lambda(\hat{y}_l), \lambda(\hat{y}_r)) \mid f \in [1, d] \wedge \exists (\mathbf{x}, y) \in \mathcal{D} : x_f = v \wedge$

$$\hat{y}_l, \hat{y}_r = \underset{y_l, y_r}{\operatorname{argmin}} \sum_{(\mathbf{x}, y) \in \mathcal{D}_l(f, v, A)} \ell(y_l, y) + \sum_{(\mathbf{x}, y) \in \mathcal{D}_r(f, v, A)} \ell(y_r, y) + \sum_{(\mathbf{x}, y) \in \mathcal{D}_u(f, v, A)} \max\{\ell(y_l, y), \ell(y_r, y)\}$$

subject to \mathcal{C}

}

▷ Select the candidate node $\hat{t} \in \mathcal{N}$ which minimizes the loss \mathcal{L}^A on the training data \mathcal{D} 3: $\hat{t} = \operatorname{argmin}_{t \in \mathcal{N}} \mathcal{L}^A(t, \mathcal{D}) = \sigma(f, v, \lambda(\hat{y}_l), \lambda(\hat{y}_r))$

▷ Robust Splitting (see Definition 3)

4: $\mathcal{D}_l \leftarrow \mathcal{D}_L(\hat{t}, A)$ 5: $\mathcal{D}_r \leftarrow \mathcal{D}_R(\hat{t}, A)$

▷ Constraint Propagation and Generation (see Definition 5)

6: $\mathcal{C}_l \leftarrow \mathcal{C}_L(\hat{t}, A)$ 7: $\mathcal{C}_r \leftarrow \mathcal{C}_R(\hat{t}, A)$ 8: **return** $\hat{t}, \mathcal{D}_l, \mathcal{D}_r, \mathcal{C}_l, \mathcal{C}_r$

TABLE II
MAIN STATISTICS OF THE DATASETS USED IN OUR EXPERIMENTS.

	census	wine	credit
n. of instances	45,222	6,497	30,000
n. of features	13	12	24
class distribution (pos.÷neg. %)	25÷75	63÷37	22÷78

B. Datasets and Threat Models

We perform our experimental evaluation on three datasets: (i) Census Income,¹² (ii) Wine Quality,¹³ and (iii) Default of Credit Cards.¹⁴ In the following, we refer to such datasets as *census*, *wine*, and *credit*, respectively. Their main statistics are shown in Table II; notice that each dataset is associated with a binary classification task.¹⁵

We therefore design three different threat models by means of a set of rewriting rules indicating the attacker capabilities, with each set tailored to a given dataset. The features targeted by those rules have been selected after a preliminary data exploration stage, where we investigated the importance and data distribution of all the features.

In the case of *census*, we define six rewriting rules: (i) if a citizen never worked, he can pretend that he actually works without pay; (ii) if a citizen is divorced or separated, he can pretend that he never got married; (iii) a citizen can present his occupation as a generic “other service”; (iv) a citizen can cheat on his education level by lowering it by 1; (v) a citizen can add up to \$2,000 to his capital gain; (vi) a citizen can add up to 4 hrs per week to his working hours. We let (i),(ii),

and (iii) cost 1, (iv) cost 20, (v) cost 50, and finally (vi) cost 100 budget units. We consider 30, 60, 90, and 120 as possible values for the budget.

In the case of *wine*, we specify four rewriting rules: (i) the alcohol level can be increased by 0.5% if its original value is less than 11%; (ii) the residual sugar can be decreased by 0.25 g/L if it is already greater than or 2 g/L; (iii) the volatile acidity can be reduced by 0.1 g/L if it is already greater than 0.25 g/L; (iv) free sulfur dioxide reduced by -2 g/L if it is already greater than 25 g/L. We let (i) cost 20, (ii) and (iii) cost 30, and (iv) cost 50 budget units. We consider 20, 40, 60, 80, 100, and 120 as possible values for the budget.

For *credit*, the attacker is represented by three rewriting rules: (i) the repayment status on August or September can be reduced by 1 month if the payment is delayed up to 5 months; (ii) the amount of bill statement in September can be decreased by 4,000 NT dollars if it is between 20,000 and 500,000; and (iii) the amount of given credit can be increased by 20,000 NT dollars if it is below 200,000. For each rule, a cost of 10 budget units is required. We consider 10, 30, 40, and 60 as possible budget values.

C. Experimental Results

We discuss below the three questions stated in Section V-A.

1) *Attacking standard decision tree ensembles:* In Figure 2, we show how the accuracy, F1, and ROC AUC of standard ensembles of decision trees trained by RF and GBDT change in presence of attacks. The x -axis indicates the testing budget of the attacker, normalized in the range $[0, 1]$, with a value of 0 denoting the unattacked scenario.

Two main findings appear from the plots. First, both GBDT and RF are severely impacted when they are attacked, and their performance deteriorates to the point of turning them into almost random classifiers already when the attacker spends just half of the maximum budget, e.g., in the case of the

¹²<https://archive.ics.uci.edu/ml/datasets/census+income>¹³<https://www.kaggle.com/c/uci-wine-quality-dataset/data>¹⁴<https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients>¹⁵The *wine* dataset was originally conceived for a multiclass classification problem; we turned that into a binary one, where the positive class identifies good-quality wines (i.e., those whose quality is at least 6, on a 0-10 scale) and the negative class contains the remaining instances.

TABLE III
COMPARISON OF ADVERSARIAL LEARNING TECHNIQUES TRAINED AND ATTACKED UNDER THE SAME BUDGET. THE TABLE ALSO SHOWS THE PERFORMANCE DIFFERENCE BETWEEN RF-TREANT AND THE BEST COMPETITOR.

		AB			RT			RF-TREANT						
		Accuracy	F_1	ROC AUC	Accuracy	F_1	ROC AUC	Accuracy		F_1		ROC AUC		
census	Budget	30	0.850	0.783	0.902	0.813	0.692	0.883	0.850	+0.0%	0.773	-1.3%	0.897	-0.6%
		60	0.783	0.690	0.827	0.810	0.698	0.871	0.845	+4.3%	0.766	+9.7%	0.894	+2.6%
		90	0.798	0.705	0.825	0.775	0.607	0.855	0.845	+5.9%	0.769	+9.1%	0.893	+4.4%
		120	0.788	0.694	0.793	0.744	0.558	0.528	0.842	+6.9%	0.762	+9.8%	0.887	+11.9%
wine	Budget	20	0.762	0.737	0.824	0.734	0.703	0.795	0.764	+0.3%	0.739	+0.3%	0.821	-0.4%
		40	0.723	0.689	0.788	0.623	0.548	0.662	0.728	+0.7%	0.689	+0.0%	0.802	+1.8%
		60	0.718	0.687	0.788	0.552	0.418	0.522	0.720	+0.3%	0.680	-1.0%	0.798	+1.3%
		80	0.715	0.680	0.773	0.566	0.443	0.561	0.728	+1.8%	0.688	+1.2%	0.800	+3.5%
		100	0.702	0.668	0.761	0.559	0.429	0.553	0.727	+3.6%	0.687	+2.8%	0.796	+4.6%
credit	Budget	10	0.811	0.644	0.749	0.799	0.610	0.748	0.816	+0.6%	0.656	+1.9%	0.765	+2.1%
		30	0.786	0.544	0.661	0.763	0.457	0.655	0.810	+3.1%	0.617	+13.4%	0.745	+12.7%
		40	0.784	0.554	0.660	0.759	0.438	0.632	0.808	+3.1%	0.618	+11.6%	0.744	+12.7%
		60	0.777	0.533	0.622	0.759	0.436	0.613	0.809	+4.1%	0.616	+15.6%	0.744	+19.6%

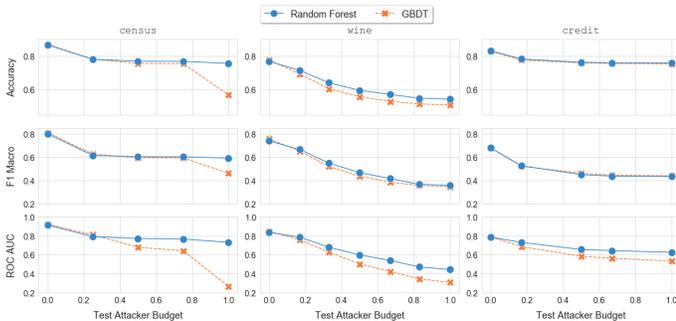


Fig. 2. The impact of the attacker on RF and GBDT.

wine dataset. On that dataset, the drop of ROC AUC ranges from -25.8% to -40.6% for GBDT and from -15.5% to -28.4% for RF, when the attacker is supplied just half of the budget. Second, RF typically behaves better than GBDT on all the validity measures, with a few cases where the improvement is very significant. A possible explanation of this phenomenon is that RF usually exhibits better generalization performance, while GBDT is known to be more susceptible to jiggling data, therefore more likely to overfit [36]. Since robustness to adversarial examples in a way resembles the ability of a model to generalize, RF is less affected by the attacker than GBDT. Still, the performance drop under attack is so massive even for RF that none of the traditional methods can be reliably adopted in an adversarial setting.

The higher resiliency of RF to adversarial examples motivated our choice to deploy TREANT on top of such ensemble method in our implementation. It is worth remarking though that TREANT is still general enough to be plugged into other frameworks for ensemble tree learning.

2) *Robustness of adversarial learning techniques:* We now measure the benefit of using adversarial learning techniques to contrast the impact of evasion attacks at test time. More specifically, we validate the robustness of our method in

comparison with the two state-of-the-art adversarial learning methods Adversarial Boosting (AB) and Robust Trees (RT). Note that the authors of [35] did not experimentally compare RT against AB in their original work.

We first investigate how robust a model is when it is targeted by an attacker with a test budget exactly matching the training budget. This simulates the desirable scenario where the threat model was defined accurately, i.e., each model is trained knowing the actual attacker capabilities. Table III shows the results obtained by the different adversarial learning techniques for the different training/test budgets. It is clear how our method outperforms its competitors, basically for all measures and datasets. Most importantly, the superiority of our approach becomes even more pronounced as the strength of the attacker grows. For example, the percentage improvement in ROC AUC over AB on the `credit` dataset amounts to 2.1% for budget 10, while this improvement grows to 19.6% for budget 60. It is also worth noticing that the performance of RT is consistently worse than that of AB.

The second analysis we carry out considers the case of adversarial learning techniques trained with the maximum available budget. We use security evaluation curves to measure how the performance of the compared methods changes when the test budget given to the attacker increases up to the maximum available. The results are shown in Figure 3, where we normalized the test budget in the range $[0, 1]$.

Two main comments can be made from the plots. First, our method constantly outperforms its competitors on all datasets and measures, especially when the attacker gets stronger. The price to pay for this increased protection is just a slight performance degradation in the unattacked setting, which is largely compensated under attack. Indeed, the performance of our method is nearly constant and insensitive to variations in the attacker's budget, which is extremely useful when such information is hard to quantify exactly. Second, we observe that AB is usually more robust than RT. We believe that RT

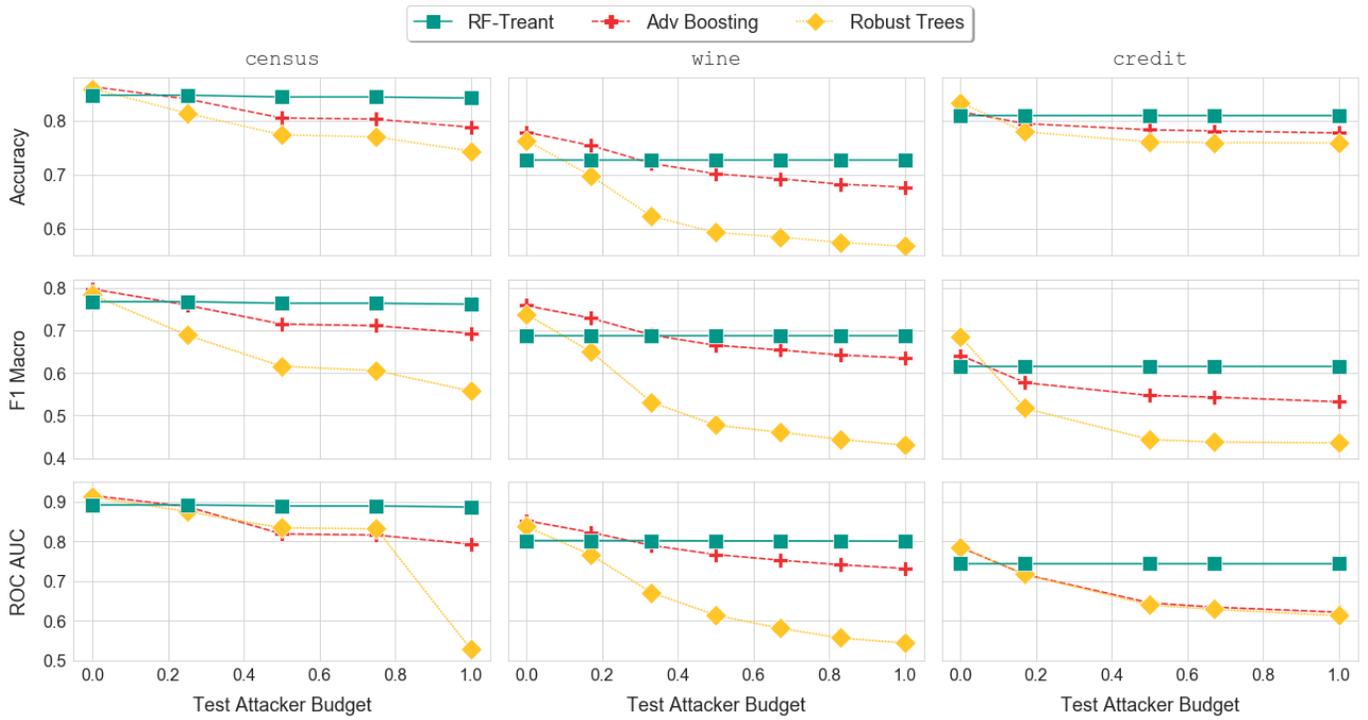


Fig. 3. Comparison of adversarial learning techniques for different test budgets and maximum train budget.

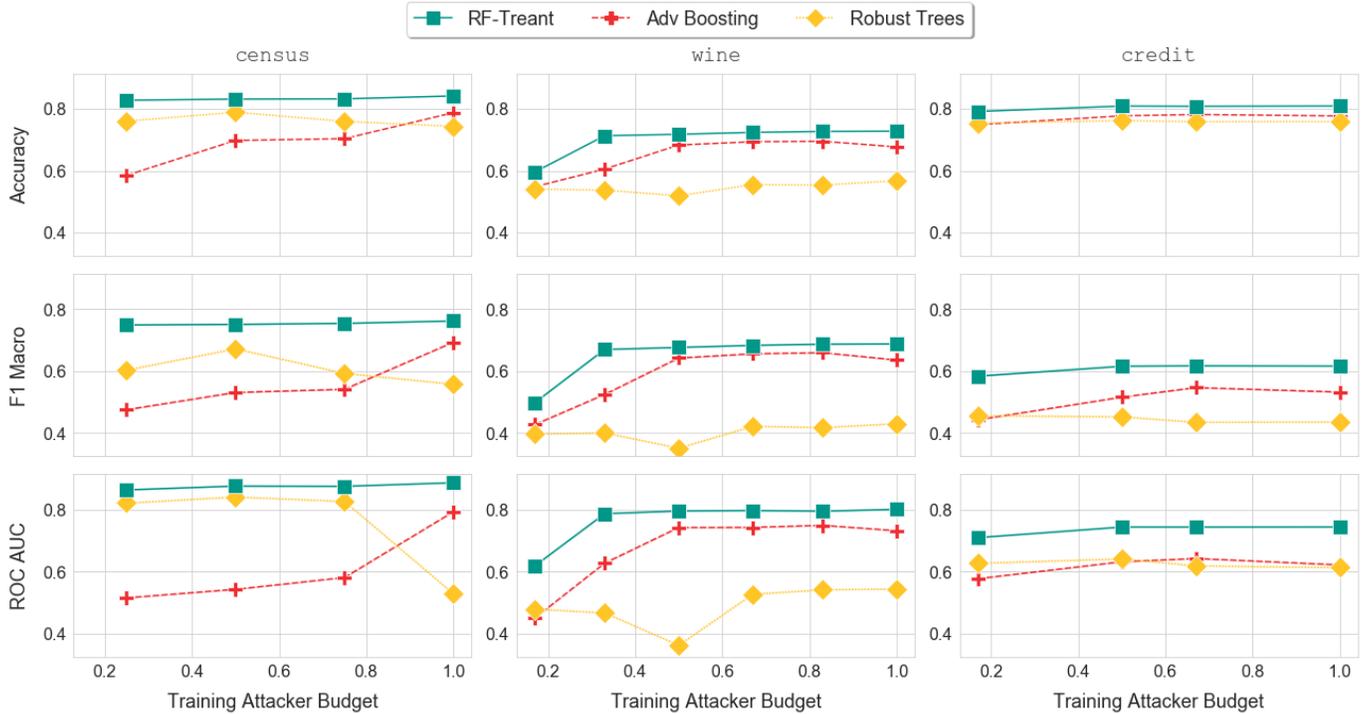


Fig. 4. Comparison of adversarial learning techniques for different train budgets and maximum test budget.

suffers from its heuristic splitting strategy, which is not smart enough to counteract the full spectrum of possible attacks, and the lack of attack invariance. There are indeed a few cases where the performance of RT upon attack is comparable to

the performance of traditional GBDT.

3) *Impact of training budget:* A last intriguing aspect to consider is how much adversarial learning techniques are affected by the assumptions made on the attacker's capabilities

upon learning, i.e., the training budget. Figure 4 is essentially the dual of Figure 3, where we consider the strongest possible attacker (with the largest test budget) and we analyze how much models learned with different (smaller or equal) training budgets are able to respond to evasion attempts.

We draw the following observations. First, our method leads to the most robust models for all measures and datasets, irrespective of the budget used for training. Moreover, our method is the one which most evidently presents a healthy, expected trend: the greater the training budget used to learn the model, the better its performance under attack. This trend eventually reaches its peak when the training budget matches the test budget. Also AB shows a similar trend, yet it suffers from a slow start before reaching its best performance, which is however worse than our method. RT is the method which shows the most unpredictable behavior, as its performance fluctuates up and down, and sometimes suddenly drops. This is likely due to the fact that the heuristic it implements is too shortsighted with respect to the set of all the attacks and the lack of attack invariance. Finally, we remark a last appealing, distinctive aspect of our method: even when the training uses a significantly smaller budget than the one used by the attacker at test time, it already achieves nearly optimal performances. The same is not true for its competitors, which complicates their deployment in real-world settings.

VI. CONCLUSION

This paper proposes TREANT, a new adversarial learning algorithm that is able to grow decision trees that are resilient against evasion attacks. TREANT is the first algorithm which greedily, yet soundly, minimizes an evasion-aware loss function, which captures the attacker’s goal of maximizing prediction errors. Our experiments, conducted on three publicly available datasets, confirm that TREANT produces accurate tree ensembles, which are extremely robust against evasion attacks. Compared to the state of the art, TREANT exhibits a ROC AUC improvement against the strongest attacker ranging from $\approx 10\%$ to $\approx 20\%$.

As future work, we plan to revise our decision tree construction to make it aware of its deployment inside an ensemble; in other words, we aim at exploiting the information that the currently grown ensemble is particularly strong or weak against some classes of attacks to guide the construction of the next member of the ensemble. We also plan to evaluate our learning technique against regression datasets to get an additional quantitative evaluation of its security benefits. Finally, we want to investigate the combined use of standard decision trees and decision trees trained using TREANT in the same ensemble, to achieve the optimal trade-off between accuracy in the unattacked setting and resilience to attacks.

REFERENCES

- [1] L. Huang, A. D. Joseph, B. Nelson, B. I. P. Rubinstein, and J. D. Tygar, “Adversarial machine learning,” in *AISeC*, 2011, pp. 43–58.
- [2] B. Biggio and F. Roli, “Wild patterns: Ten years after the rise of adversarial machine learning,” *Pattern Recognition*, vol. 84, pp. 317–331, 2018.
- [3] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Srndic, P. Laskov, G. Giacinto, and F. Roli, “Evasion attacks against machine learning at test time,” in *ECML PKDD*, 2013, pp. 387–402.
- [4] A. M. Nguyen, J. Yosinski, and J. Clune, “Deep neural networks are easily fooled: High confidence predictions for unrecognizable images,” in *CVPR*, 2015, pp. 427–436.
- [5] N. Papernot, P. D. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, “The limitations of deep learning in adversarial settings,” in *EuroS&P*, 2016, pp. 372–387.
- [6] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, “Deepfool: A simple and accurate method to fool deep neural networks,” in *CVPR*, 2016, pp. 2574–2582.
- [7] D. Lowd and C. Meek, “Adversarial learning,” in *SIGKDD*, 2005, pp. 641–647.
- [8] B. Biggio, B. Nelson, and P. Laskov, “Support vector machines under adversarial label noise,” in *ACML*, 2011, pp. 97–112.
- [9] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” in *ICLR*, 2014.
- [10] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” in *ICLR*, 2015.
- [11] F. Chollet, *Deep Learning with Python*, 1st ed. Greenwich, CT, USA: Manning Publications Co., 2017.
- [12] G. Tolomei, F. Silvestri, A. Haines, and M. Lalmas, “Interpretable predictions of tree-based ensembles via actionable feature tweaking,” in *SIGKDD*, 2017, pp. 465–474.
- [13] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” in *ICLR*, 2018.
- [14] E. B. Hunt, J. Marin, and P. J. Stone, “Experiments in induction,” 1966.
- [15] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
- [16] S. Hershkop and S. J. Stolfo, “Combining email models for false positive reduction,” in *SIGKDD*, 2005, pp. 98–107.
- [17] R. Perdisci, G. Gu, and W. Lee, “Using an ensemble of one-class SVM classifiers to harden payload-based anomaly detection systems,” in *ICDM*, 2006, pp. 488–498.
- [18] T. P. Tran, P. Tsai, and T. Jan, “An adjustable combination of linear regression and modified probabilistic neural network for anti-spam filtering,” in *ICPR*, 2008, pp. 1–4.
- [19] B. Biggio, G. Fumera, and F. Roli, “Multiple classifier systems for robust classifier design in adversarial environments,” *Int. J. Machine Learning & Cybernetics*, vol. 1, no. 1-4, pp. 27–41, 2010.
- [20] W. He, J. Wei, X. Chen, N. Carlini, and D. Song, “Adversarial example defense: Ensembles of weak defenses are not strong,” in *WOOT*, 2017.
- [21] V. Vapnik, “Principles of risk minimization for learning theory,” in *Advances in neural information processing systems*, 1992, pp. 831–838.
- [22] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Wadsworth, 1984.
- [23] L. Hyafil and R. L. Rivest, “Constructing optimal binary decision trees is np-complete,” *Inf. Process. Lett.*, vol. 5, no. 1, pp. 15–17, 1976.
- [24] S. K. Murthy, “Automatic construction of decision trees from data: A multi-disciplinary survey,” *Data Min. Knowl. Discov.*, vol. 2, no. 4, pp. 345–389, 1998.
- [25] J. H. Friedman, “Greedy function approximation: a gradient boosting machine,” *Annals of statistics*, pp. 1189–1232, 2001.
- [26] B. Nelson, B. I. P. Rubinstein, L. Huang, A. D. Joseph, S. Lau, S. J. Lee, S. Rao, A. Tran, and J. D. Tygar, “Near-optimal evasion of convex-inducing classifiers,” in *AISTATS*, 2010, pp. 549–556.
- [27] B. Biggio, G. Fumera, and F. Roli, “Security evaluation of pattern classifiers under attack,” *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 4, pp. 984–996, 2014.
- [28] N. Srndic and P. Laskov, “Practical evasion of a learning-based classifier: A case study,” in *S&P*, 2014, pp. 197–211.
- [29] A. Kantchelian, J. D. Tygar, and A. D. Joseph, “Evasion and hardening of tree ensemble classifiers,” in *ICML*, 2016, pp. 2387–2396.
- [30] N. Carlini and D. A. Wagner, “Towards evaluating the robustness of neural networks,” in *S&P*, 2017, pp. 39–57.
- [31] H. Dang, Y. Huang, and E. Chang, “Evasion classifiers by morphing in the dark,” in *CCS*, 2017, pp. 119–133.
- [32] H. Xiao, B. Biggio, B. Nelson, H. Xiao, C. Eckert, and F. Roli, “Support vector machines under adversarial label contamination,” *Neurocomputing*, vol. 160, pp. 53–62, 2015.
- [33] S. Gu and L. Rigazio, “Towards deep neural network architectures robust to adversarial examples,” in *ICLR, Workshop Track Proceedings*, 2015.

- [34] N. Papernot, P. D. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *S&P*, 2016, pp. 582–597.
- [35] H. Chen, H. Zhang, D. S. Boning, and C. Hsieh, "Robust decision trees against adversarial examples," in *ICML*, 2019, pp. 1122–1131.
- [36] S. Nawar and A. Mouazen, "Comparison between random forests, artificial neural networks and gradient boosted machines methods of on-line vis-nir spectroscopy measurements of soil total nitrogen and total carbon," *Sensors*, vol. 17, no. 10, p. 2428, 2017.