Using Randomness to Improve Robustness of Tree-Based Models Against Evasion Attacks

Fan Yang, Zhiyuan Chen, and Aryya Gangopadhyay

Abstract—Machine learning models have been widely used in security applications. However, it is well-known that adversaries can adapt their attacks to evade detection. There has been some work on making machine learning models more robust to such attacks. However, one simple but promising approach called *randomization* is under-explored. In addition, most existing works focus on models with differentiable error functions while tree-based models do not have such error functions but are quite popular because they are easy to interpret. This paper proposes a novel randomization-based approach to improve robustness of tree-based models against evasion attacks. The proposed approach incorporates randomization into both model training time and model application time (meaning when the model is used to detect attacks). We also apply this approach to random forest, an existing ML method which already has incorporated randomness at training time but still often fails to generate robust models. We proposed a novel weighted-random-forest method to generate more robust models and a clustering method to add randomness at model application time. We also proposed a theoretical framework to provide a lower bound for adversaries' effort. Experiments on intrusion detection and spam filtering data show that our approach further improves robustness of random-forest method.

Index Terms—Evasion attacks, machine learning, adversarial learning, intrusion detection, spam filtering

1 Introduction

With the arrival of big data era, machine learning techniques have been widely used to build models for cyber security applications such as spam filtering [1], [2], malware or virus detection [3], and intrusion detection [4], [5], [6], [7]. However, attackers may use a type of attacking strategy called *evasion attack* which modifies their data to avoid detection. For example, an email spammer may modify spam emails to drop or add certain words or symbols and a hacker can modify the signature of a malware or virus.

There have been studies on vulnerability of AI/ML models [8], especially more recently on deep learning models [9], [10], [11], [12], [13], [14]. There also has been work on building more robust mining models against evasion attacks [8], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26]. Most existing works focus on models with differentiable error functions because attackers can use techniques such as gradient descent to decide the modification to original instance which will lead to maximal error. However tree-based models (e.g., decision trees, regression trees, random forest) do not have such error functions but are quite popular because they are easy to interpret.

In addition, most existing studies use deterministic models. However for tree-based models, a deterministic model should be concise to avoid overfitting the training data according to the Minimum Description Length (MDL) principle [27]. A concise model often uses a small number of features or a small number of features may have much larger impact on the output than other features. Attackers can modify such features to evade detection. To understand the problem of deterministic models, let us look at the following example.

Example 1. Let us consider an email spam filtering example. Suppose the filtering software uses decision trees. Fig. 1 shows several sample decision trees (in practice decision trees will have more nodes but here we simplify the trees to show the concept). Each node represents fraction of a word or an email in content of an email except for "total capital", which represents the total number of capitalized letters in the email. The numbers in parenthesis are number of positive or negative training instances in each node.

If only one decision tree, say f_1 is used, it is very easy for attackers to modify a spam email to evade detection. For example, suppose attackers have an email with feature values shown in Table 1(a). Attackers may learn that the spam filtering software looks at the "remove" and "$" features. As a result attackers just need to modify one feature (the percentage of $ sign in the email) to avoid detection.

Some researchers try to use an ensemble approach (i.e., build a number of models instead of one) [28], [29] to improve robustness of models. However, although the ensemble approach does add some uncertainty to the generated models, it has two shortcomings: 1) its goal is still to detect original non-evasive attacks, so the models built by ensemble approach may still frequently use a small subset of features, making it vulnerable to evasion attacks; 2) ensemble approach is still deterministic at model application time, making it easier for attackers to adapt.

• *The authors are with the Department of Information Systems, University of Maryland Baltimore County, Baltimore, MD 21250 USA. E-mail: {fyang1, zhchen, gangopad}@umbc.edu.*

Manuscript received 3 June 2019; revised 26 Feb. 2020; accepted 6 Apr. 2020. Date of publication 15 Apr. 2020; date of current version 11 Jan. 2022. (Corresponding author: Fan Yang.) Recommended for acceptance by J. Chen. Digital Object Identifier no. 10.1109/TKDE.2020.2987299

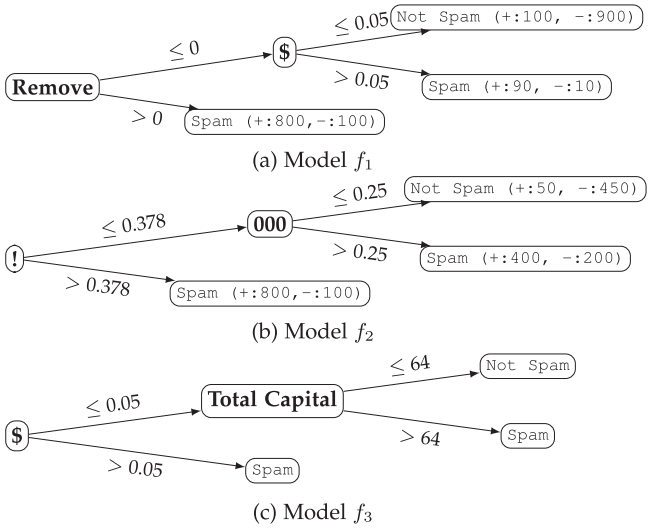


Fig. 1. Decision tree models created by our model in Example 1.

TABLE 1
Some Possible Attacks for Example 1

Remove	$	!	000	Total Capital
0	0.2	0.4	0.3	100

(a) Attacker's original spam email

Remove	$	!	000	Total Capital
0	**0.05**	0.4	0.3	100

(b) Modified email to trick f_1 (bold as changes)

Remove	$	!	000	Total Capital
0	**0.05**	0.4	0.3	**64**

(c) Modified email to trick at least two models in Figure 1

Remove	$	!	000	Total Capital
0	**0.05**	**0.378**	**0.25**	100

(d) Modified email to trick f_1 and f_2

Random forest is a tree-based ensemble method which generates a pool of decision trees. Each tree is trained using a random sample of training data (this is also called *bagging*) and a random subset of features at each split (also called *random subspace* or *feature bagging*). However we found random forest often fails to generate robust models. For example, We ran random forest on Spambase data set [30], which is an e-mail spam data set with 57 features and built 100 decision trees. Fig. 2 reports for each feature (attribute), the number of trees using that feature. 9 out of 57 features appear in at least 80 percent of trees and 22 features appear in at least half of trees. This means that attackers can modify these frequently used features to change prediction outcome.

Our Contributions. This paper proposes a weighted random forest approach to generate a more robust pool of decision trees at model training time as well as a clustering-based method to add randomness at model application time. Next we use an example to show how our approach may help.

In Example 1, suppose our method builds a pool of three decision trees shown in Fig. 1. Table 1(c) shows the number of features an attacker needs to modify if all three trees are used at model application time (i.e., at least two trees need to return "not spam"). Attackers need to modify at least two features now (using f_1 alone just needs one modification).

Our method further boosts robustness at model application time. For example, if all three trees are used in Fig. 1. As shown in Table 1(c), attackers only need to modify two features to avoid detection. However if we select f_1 and f_2 (or f_2 and f_3) at model application time and the spam filtering software will only label the email "not spam" if both trees return "not spam", attackers have to modify at least three features as shown in Table 1(d).

This paper makes the following contributions:

1) Methods to build a diverse pool of mining models for tree-based methods. Tree-based methods don not have differentiable error functions. Unlike methods based on retraining using adversarial examples [31], the proposed methods modified the random forest algorithm to generate more robust trees.
2) A method to randomly select a subset of models at model application time to further boost robustness. To the best of our knowledge, this is the first paper proposing this method.
3) A theoretical framework that provides a lower bound to adversaries' effort.
4) Experiments to compare our methods to existing methods. We also showed how to set parameters in our methods to optimize the defense against both non-evasive attacks and evasion attacks.

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 introduces some background information and gives an overview of our approach. Section 4 describes how our approach can be applied to random forest. Section 5 presents a theoretical framework to bound adversaries' cost. Section 6 presents experimental results. Section 7 discusses how our approach can be extended to other mining models and Section 8 concludes the paper.

2 Related Work

Next we briefly describe related work in the literature, which can be roughly divided into five categories.

The first category of work studies attacks against conventional mining models [32]. Barreno *et al.* described a taxonomy of attacks and briefly mentions a few possible defense strategies [16]. Lowd and Meek [8] studied a reverse engineering algorithm to learn the models of a given machining learning algorithm. They assume that the adversary can find out the outcome of a prediction model by sending instances to the model. Attacks on intrusion detection are discussed in [33]. Nelson et. al. [34] studied attacks against classifiers with convex decision boundaries. Tramer *et al.* studied attacks that steal machine learning models based on output of such models [35].

The second category of work tries to find robust learning algorithms in presence of attacks. A common approach is to model the learning problem as an optimization problem [36], [37], [38]. Globerson and Roweis [21] studied the problem of building a robust SVM classifier in presence of feature deletion attacks (attackers can delete up to a certain number of features from the test data). Another adversarial

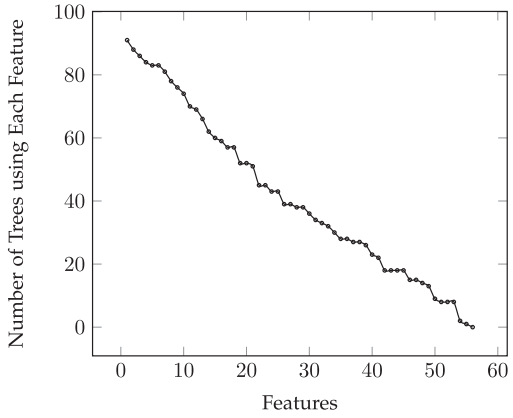


Fig. 2. Number of trees using each feature by random forest on Spambase data set.

learning method is proposed for SVM in [39], which considers two possible settings for attackers, in one of them the attackers can corrupt data without any restriction and in the other one the attackers have costs associated with attacks.

The third category of work applies game theory to the adversarial learning problem. Typically the problem is modeled as a two-player and multi-stage game between the data miner and the adversary. At each stage, the adversary tries to find the best possible attacks and the data miner adjusts the mining models to such attacks. Kearns and Li [40] proposed a theoretical upper bound on tolerable malicious error rates. Dalvi *et al.* [41] proposed a game theory framework which models the data miner and the adversary as a two player game. They assume that both players have perfect knowledge (i.e., data miner knows the adversary's attacking strategy and the adversary knows the mining model). Several other works [19], [42], [43] model the adversarial learning problem as a Stackelberg game.

The fourth category of work focuses on vulnerabilities of deep neural nets [10], [11], [12], [13], [14]. A survey can be found at [9]. Most of these studies focus on image classification and they have shown that adversarial examples can effectively fool a neural network to misclassify a slightly modified image. There has been some effort to make deep neural nets more robust [24], [25], [26], where most of them retrain the neural nets with added adversarial examples. However most of such work still focuses on images. Image-based methods operate in a continuous feature space so they may not be directly applicable to cyber security applications which contain a lot of discrete attributes (e.g., words in emails or network protocol for intrusion detection data). To the best of our knowledge, the only exception is [12], where Grosse *et al.* have studied how to generate adversarial examples for malware classification and use these examples to retrain a deep neural net. In addition, neural nets have differentiable error functions but tree-based methods do not.

The final category of work use ensemble methods in adversarial settings [44], [45], [46], [47], [48]. For example, Biggio *et al.* [29] used two methods: random subspace (a random subset of features are used for training each model) and bagging (a random sample of training data is used for training each model). Although these methods are similar to our approach, they do not consider the problem of optimizing defense against both evasion and non-evasive attacks (original attacks), which is the focus of our approach. These solutions also only consider the model building time, and our approach will inject randomness into model application time as well.

Kantchelian *et al.* modeled the problem of finding optimal evasion for tree ensemble classifiers (including random forest) as a mixed integer linear programming problem [31] and proposed to improve robustness by augmenting the training set with evading instances. However, this approach has several problems: 1) it still runs a conventional random forest algorithm on augmented training set; our experiments will show that there are still features vulnerable to evasion attacks after retraining; 2) it is quite expensive to solve the mixed integer linear programming problem.

Overall there has been very little work using randomization at model application time. The only work we are aware of that uses randomization at model application time is [49], where the authors considered an optimal strategy when the system can use several classifiers. They found the optimal solution is either to choose a classifier uniformly at random or choose the classifier with the smallest error depending on the relative importance between accuracy versus robustness. However, this work does not consider how to create these classifiers and does not test their approach on real data sets. We propose a method to build more diverse pool of models and a clustering-based method to select these models at model application time. We also test our solution on real data sets.

To summarize, our work differs from existing work on two aspects: 1) our approach adds randomness at model application time; 2) our approach considers tree-based methods, which do not have differentiable error function.

3 Background and Overview of Our Approach

We first introduce some notations. Let $\mathcal{L}$ be a data mining algorithm which given an instance x_i returns a class label (malicious or benign). Let $T = \{(x_i, y_i)\}_{i=1}^{N}$ be a set of N training samples where $x_i = (x_{i1}, \ldots, x_{im})$ is a training instance with m independent variables (or features) $A_1, \ldots, A_m$ and x_{ij} is the value of A_j in x_i, and y_i is the value of dependent variable Y for x_i. Let M be the number of models in the model pool. We will build a pool $\mathcal{P} = \{f_1(x), \ldots, f_M(x)\}$ of prediction models where each $f_i(x)$ is a model to predict the value of Y for record x. At model application time, we will select a subset of $\mathcal{P}$ for prediction.

Threat Model. This paper focuses on evasion attacks where attackers can observe the prediction outcome of the detection model and modify their attacking instances accordingly to avoid detection, but cannot tamper with the model and training data directly (also called *exploratory integrity attacks* in [16]).

Following the literature, We will consider two cases for attackers' knowledge.

Definition 1. *White-box attack: Attackers know the models being built over training data and can modify their attacking instances based on that.*

Definition 2. *Black-box attack: Attackers do not know the models being built over training data, but can observe the predictions*

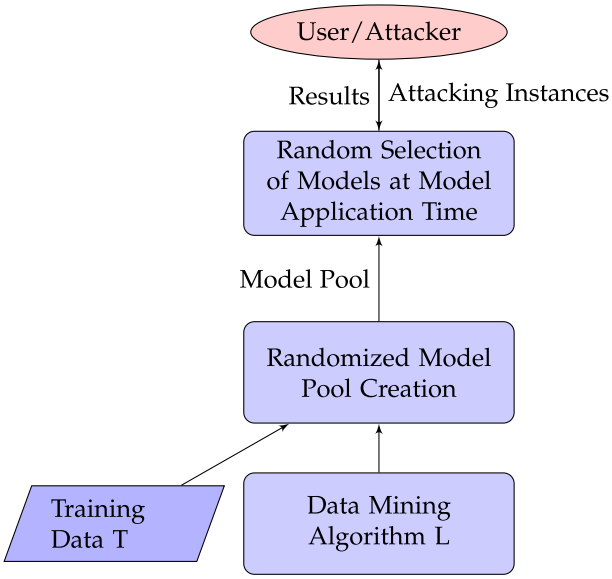


Fig. 3. Architecture of our approach.

made by the models and have a sample data set similar to those used in training.

The black-box case is more common in practice. The white-box case often occurs if attackers are insiders.

In the literature the robustness of a model is often measured by the amount of effort attackers need to pay to evade a model. This is often achieved by solving the following problem for attackers.

Definition 3. *Optimal Evasion Problem: Given a malicious instance x_i that is initially labeled as malicious by the learning algorithm $\mathcal{L}$, attackers want to find an instance x_i' such that 1) $\mathcal{L}$ will classify it as benign; 2) the distance $d(x_i, x_i')$ between x_i and x_i' is minimized.*

In the literature L_p distance is often used as distance function. In this paper we use weighted L_p distance

$$d(x_i, x_i') = \left(\sum_{j=1}^{m} c_j |x_{ij} - x_{ij}'|^p \right)^{1/p}. \quad (1)$$

Here c_j is a weight that represents the difficulty or cost of modifying feature j. The modified instance x_i' that can evade detection is also called adversarial examples/instances according to the literature. A model is more robust to evasion attacks if attackers need to put more effort to generate adversarial examples (i.e., with larger L_p distances to original instances).

The detection model need to consider both evasion attacks and original (unmodified) non-evasive attacks because in practice both types of attacks may occur. There is usually a tradeoff between defense against these two types attacks as a model that is good at detecting one type of attack may have poor detection rate against the other type of attack. E.g., random forest is good at detecting non-evasive attacks but is bad at evasion attacks. As another example, a model that labels more instances as attacks may be robust to evasion attacks but will have low precision for original attacks as many normal instances may get labeled as attacks. So the goal of this paper is to optimize this tradeoff.

Fig. 3 shows the architecture of our approach. Given a training data set T and a data mining algorithm L, our approach first generates a model pool using randomization. When user provides an instance z to classify, a random selection process will be used to select a subset of models from the model pool to return a prediction for z. Next we show how our approach can be applied to random forest method.

4 Our Solution for Random-Forest

As illustrated in Section 1, random forest often fails to generate robust pool of models. We developed a Weighted Random Forest algorithm (Section 4.1) to address this issue. The key idea is to penalize features with high vulnerability (or more frequent appearance) such that features are used more uniformly in different trees. We also propose a clustering based method to add randomness at model application time (Section 4.2).

4.1 Weighted Random Forest Method

Algorithm 1 shows the pseudo code of the Weighted Random Forest (WRF) algorithm. The algorithm is the same as random forest except that when the algorithm selects a feature to split, the splitting criteria of each feature A_j is multiplied by a weight w_j. This weight will be used to optimize the tradeoff between defense against non-evasive attacks (goal of the original random forest algorithm) and evasion attacks.

Algorithm 1. Weighted Random Forest Method

Input: Training data set $T = \{(x_i, y_i)\}_{i=1}^{N}$, number of models M, feature subset size F
Output: A set of candidate models $f_1, \ldots, f_M$
1 Compute weights $w_1, \ldots, w_m$ for each feature
2 **for** $i = 1$ *to* M **do**
3 Draw a uniform random sample with replacement of size N from T, let the sample be T_i
4 Build a decision tree f_i on T_i where at each node a random subset of F features are used and the splitting criteria for each feature A_j is multiplied by weight w_j
5 **end**
6 **return** $f_1, \ldots, f_M$

We compute the weights based on the following two observations. First, features that are more difficult to change (i.e., with larger c_j in Equation (1)) should receive higher weights. Second, if modifying a feature is likely to change the classification outcome of a positive (malicious) instance to negative (benign), then this feature should have a lower weight. We call such features *vulnerable features* and proposed two metrics to quantity vulnerability. The first is information gain and the the other is a new metric called *differential ratio*.

Algorithm 2 shows the steps to compute weights. Line 1 runs standard random forest algorithm on the training data to generate an initial set $\mathcal{P}_0$ of M trees. These trees will be used to compute feature weights. For each feature A_j and each tree f_i that uses A_j, the algorithm computes differential ratio or information gain at each node using A_j as splitting node. Since the same feature may be used multiple times in the same tree, the algorithm computes r_{ij} as the maximal information gain or differential ratio over all such appearance

in tree f_i. Finally line 6 computes weight w_j for feature A_j using all r_{ij}.

Algorithm 2. Compute Feature Weights

Input: Training data set $T = \{(x_i, y_i)\}_{i=1}^{N}$, number of models M
Output: Weights $w_1, \ldots, w_m$ for each feature

1 Run standard random forest algorithm on T to create an initial model pool $\mathcal{P}_0$
2 **for** *each feature* A_j, $1 \le j \le m$ **do**
3 **for** *each tree* f_i *in* $\mathcal{P}_0$ *that uses* A_j **do**
4 Compute r_{ij} as the maximal information gain or differential ratio for A_j in f_i
5 **end**
6 $w_j = e^{-r\frac{\sum_{i=1}^{M} r_{ij}}{Mc_j}}$
7 **end**

Next we describe how to use information gain or differential ratio to compute the weights.

Information Gain. One observation is that a vulnerable feature is often selected as splitting attribute because it appears in many trees. Random forest selects splitting attribute based on measures such as information gain so we can use these measures to quantify vulnerability of a feature.

Let $vals(A_j)$ be set of possible values of feature A_j. Let T_n be set of training instances under node n. Let T_v be the set of training instances in T_n and have value v on feature A_j. Information gain for a node n that splits on feature A_j is defined in Equation (2)

$$IG(A_j, n) = H(T_n) - \sum_{v \in vals(A_j)} \frac{|T_v|}{|T_n|} H(T_v). \tag{2}$$

Here $H(T_n)$ and $H(T_v)$ are entropy of class distribution for node n and child node n_v, respectively.

Since a feature may appear multiple times in the same tree, we take the maximal information gain in a tree so we consider the most vulnerable case for a feature A_j (i.e., the worst case). Let f_i be a tree in the model pool $\mathcal{P}_0$ built by random forest

$$r_{ij} = \max_{n \in f_i, n \text{ splits on } A_j} IG(A_j, n). \tag{3}$$

We then sum up r_{ij} in all trees and divide the sum by M (total number of trees). We then compute weight w_j for feature A_j as

$$w(A_j) = e^{-r\frac{\sum_{i=1}^{M} r_{ij}}{Mc_j}}. \tag{4}$$

Here c_j is the difficulty of modifying A_j in Equation (1). r is a parameter to adjust the importance of robustness. If $r = 0$, $w(A_j) = 1$ for all features and our method is identical to random forest. For a positive r value, the weight of a feature increases with the difficulty of modifying that feature and decreases with the information gain. So our method favors features that are difficult to modify or are less vulnerable.

The exponential function in Equation (4) is used for smoothing. For example, suppose a feature A_1 has an information gain of 0.15 and a feature A_2 has an information gain of 0.01, and both features have c_j of 1. Without the exponential function the weight of A_1 will be 15 times of that of A_2 (both are negative weights). This may penalize feature A_1 too much because features with high information gain are features that can better distinguish positive instances from negative instances. With the exponential function the weight for A_1 is 0.55 and weight for A_2 is 0.96 when $r = 4$. We will discuss how to choose appropriate value of r in Section 6.

Differential Ratio. One problem of information gain is that it is mainly used for classification accuracy, and it may not precisely capture vulnerability of a feature at times. So we proposed an alternative metric called *differential ratio* to quantify vulnerability. We start by considering binary trees, and then explain the difference between differential ratio and information gain, and finally generalize our solution to multi-branch trees.

Let $p_+(n_l)$ be the fraction of positive training instances in the subtree rooted at node n's left child and $p_+(n_r)$ be the fraction of positive instances in the subtree rooted at its right child. Let $|n|$ be the total number of training instances in the subtree rooted at node n and $|T|$ be the total number of training instances. Let A_n be the splitting feature used at node n. We calculate a *differential ratio* for feature A_n at n as

$$d(A_n, n) = |p_+(n_l) - p_+(n_r)| \frac{|n|}{|T|}. \tag{5}$$

Next we explain the intuition behind Equation (5). Let x be a positive instance that falls under the subtree rooted at n. Modifying the splitting feature of n may change $x's$ classification outcome from positive to negative. Now we try to estimate the probability of this change. We assume that test cases will follow a similar distribution as the training cases (this is the basis for all data mining algorithms). Thus we can estimate the probability of a test case reaching node n by $\frac{|n|}{|T|}$. Since the test case is positive, it is more likely belonging to the child node with higher fraction of positive cases. Without loss of generality we assume that the left child has higher fraction of positive cases. Let A_n be the splitting feature at node n. Modifying A_n will move x from left child to right child. We use $p_+(n_l)$ to approximate the probability of x classified as positive in the left child, and $p_+(n_r)$ to approximate the same probability in right child. So the probability of x being classified as negative after modifying A_n can be estimated by $p_+(n_l)(1 - p_+(n_r))$.

However this measure has two problems: 1) it is not symmetric; 2) it is greater than zero even if the right child has higher fraction of positive instances (so moving x to right child will not help the attackers). So instead we use $|p_+(n_l) - p_+(n_r)|$ which is both symmetric and indicates that x is more vulnerable when one of its child has much higher fraction of positive instances than the other child (so moving x to the other child helps attackers).

Once we compute differential ratio for a feature A_j at a single node, we can compute the differential ratio in the whole model pool in a similar way as for information gain in Equation (3). For each tree in the model pool, we take the maximal ratio r_{ij} across all nodes splitting on A_j. We then use Equation (4) to compute the weight of A_j.

Differences Between Differential Ratio and Information Gain. There are two main differences between information gain

and differential ratio: 1) information gain considers original class distribution ($H(T_n)$), differential ratio does not; 2) information gain considers the size of each child node (the entropy of each child node is weighted by size), differential ratio does not.

Fig. 1 shows the difference between differential ratio and information gain. Suppose the upper branches are the left branches. The differential ratio for the "$" node in tree f_1 equals $|0.1 - 0.9|\frac{1100}{2000} = 0.88$, and the differential ratio for the "000" node in tree f_2 is $|0.1 - 0.67|\frac{1100}{2000} = 0.62$. So the first one has higher differential ratio. However, if we compute information gain (IG), IG for "$" node is 0.19 and IG for "000" node is 0.26. So the second node has higher IG. The reason is that information gain considers entropy before the split as well as each child node's size, while differential ratio only considers the difference of fraction of positive instances in two child nodes. Here the "$" node has one child with mostly positive instances and the other with mostly negative instances, so modifying "$" feature is more likely to change a positive instance to a negative instance. On the other hand, the higher IG for "000" is mostly due to the higher entropy before the split, which is not directly related to vulnerability.

Generalization to Multi-Branch Trees. To generalize differential ratio to multiple-branch trees, we divide children of a node n into two groups. The first group consists of child nodes with majority as positive training instances, and the second group consists of nodes with majority as negative training instances (if one of the groups is empty then differential ratio of n is zero). Let $p_+(n_+)$ be the fraction of positive instances in the first group and $p_+(n_-)$ be the fraction of positive instances in the second group. We define differential ratio for feature A_n at node n as

$$d(A_n, n) = |p_+(n_+) - p_+(n_-)|\frac{|n|}{|T|}. \tag{6}$$

We then use this differential ratio in Equations (3) and (4).

Computational Complexity. The computational complexity of random forest is $O(mMN\log N)$ where m is number of features, N is number of training instances and M is number of models. WRF builds models twice (the first pass to generate $\mathcal{P}_0$ without the weighting scheme and the second pass with the weighting scheme). Once $\mathcal{P}_0$ is generated, computing differential ratio or information gain just needs to traverse each tree in $\mathcal{P}_0$ and costs $O(\max|f|M)$ where $\max|f|$ is the maximal number of nodes in a tree. Normally $N >> \max|f|$, so the complexity of WRFD is still $O(mMN\log N)$.

4.2 Clustering-Based Model Selection at Model Application Stage

At model application stage we can dynamically select a subset of models each time the models are asked to classify an instance such that it is even harder for attackers to find out what models are used.

We propose a clustering-based model selection method (shown in Algorithm 3). This method is based on the observation that if two trees share very few common features then they should be robust to evasion attacks because attackers need to modify more features.

The algorithm first creates a similarity graph where each node is a tree in $\mathcal{P}$ and two nodes are linked if they share common features and the weight of the link is the sum of differential ratio or information gain of shared features. It then uses spectral clustering to divide the models into s clusters such that there are few between cluster links. The clustering step can be done offline. At model application time, for each test case, the clustering method randomly selects q models from each cluster. These qs models will be used to classify this test case. Note that different models will be used to classify different test cases. Since models in different clusters share very few common features, the selected models also share fewer common features than the original model pool. We will discuss how to empirically select q and s in Section 6.

Algorithm 3. Clustering-Based Model Selection Algorithm

Input: A model pool $\mathcal{P} = \{f_1, \ldots, f_M\}$, parameters s, q, and a test case t

Output: A subset of models to classify t

1 Creates a similarity graph $G = (V, E)$ where node $v \in V$ is a tree in $\mathcal{P}$ and two nodes are linked by an edge e if they share common features and e's weight is the sum of differential ratio or information gain of shared features

2 Use spectral clustering to create s clusters

3 At model application time, randomly select q models per cluster and return them

Let M be the number of trees and m be the number of features. It takes $O(mM^2)$ time to build the similarity graph. The cost of spectral clustering is $O(M^3)$. So the computational complexity of the clustering-based method is $O(M^3 + mM^2)$. Note that this cost is not related to number of instances.

The clustering-based algorithm can use models generated by our weighted (WRF) algorithm. We call the combined algorithm Cluster-based Weighted Random Forest (CWRF). It can also use models generated by a traditional random forest algorithm without feature weighting. This may make sense if users do not want to modify the implementation of random forest or do not want to pay the overhead of WRF (WRF needs to build models twice).

5 Theoretical Framework

We propose a theoretical framework to provide a lower bound to attackers' effort. We will consider L_0 distance in Equation (1). So the cost of modifying a set of features in a set SA equals $\sum_{A_j \in SA} c_j$. We start with a theoretical bound for random forest. We first introduce some notations.

Definition 4. *Let f be a decision tree to detect whether data instance x is positive or negative as a cyber security threat (e.g., a spam, an intrusion, or a fraud). A critical path of f is a root-to-leaf path p with nodes $n_1, n_2, \ldots, n_{|p|}$ where $n_{|p|}$ is a leaf node with positive label.*

For example, in the first tree f_1 in Fig. 1, there are two critical paths: {"remove" > 0}, {"remove" ≤ 0, "$" > 0.05}. It is clear that any instance labeled positive must lie on one of the critical paths (actually at the leaf node) and to turn such an instance into negative, attackers need to modify some features on the critical path.

Definition 5. *Critical count $CC(A_j)$ for a feature A_j in a set of models $\mathcal{P} = \{f_1, \ldots, f_M\}$ equals the number of trees in $\mathcal{P}$ that have A_j on at least one critical path.*

For instance, in Example 1, the critical count for "$" is two because it appears in critical paths in f_1 and f_3. The critical count for the remaining features is all one because each only appears in one tree's critical paths. Next we give the bound.

Theorem 1. *A pool $\mathcal{P} = \{f_1(x), \ldots, f_M(x)\}$ of decision trees satisfies (t_1, t_2, k)-robustness if and only if for any set SA of k features, $\sum_{A_j \in SA} CC(A_j) \leq t_1$ and $\sum_{A_j \in SA} c_j \geq t_2$. So for any positive data instance x with current positive vote (i.e., number of trees labeling x as positive) greater or equal to $\lceil M/2 \rceil + t_1$, an attacker needs to modify more than k features or needs more than t_2 L_0-distance to evade detection by $\mathcal{P}$.*

The proof is quite straightforward. Suppose x is classified as positive by a tree f. To modify x such that f will classify modified x as negative, an attacker must modify some feature on critical paths of f. Since $CC(A_j)$ is the number of trees with feature A_j on their critical paths, modifying A_j can reduce the positive vote count by at most $CC(A_j)$. Since the sum of any k features' critical count is at most t_1, the change in positive vote count by modifying k features is at most t_1. Since the current positive vote count is no less than $\lceil M/2 \rceil + t_1$, the new count is at least $\lceil M/2 \rceil$ after changing k features. Thus x will be still classified as positive. Since we need to modify more than k features to change the outcome, the L_0 distance between the adversarial instance and the original instance is at least t_2 because changing k features already results in a L_0 distance of t_2.

For example, let $\mathcal{P} = \{f_1, f_2\}$ in Fig. 1, the maximal critical count for each feature is one, so $\mathcal{P}$ satisfies $(1, c*, 1)$-robustness where $c*$ is the minimal c_j among those features. According to Theorem 1, for any instance with positive vote 2 (e.g., the original instance shown in Table 1(a)), the attacker needs to change at least 2 features to avoid detection. This bound is also tight because for a test case with feature "Remove" $= 0$, "$" $= 0.2$, "!" $= 0.2$, and "0000" $= 0.3$, attackers just need to change "$" to 0.05 and "000" to 0.25 to change the prediction of both f_1 and f_2 from spam to not spam.

Theorem 1 provides a worst-case bound. However in practice the performance is usually better because not every modification of a feature on a critical path will lead to change of classification outcome. For instance, for the first test case in Table 1(a), attackers need to modify three features instead of two for model set $\{f_1, f_2\}$ as shown in Table 1(d). The differential ratio proposed in Equation (3) can be seen as a more realistic estimation of robustness but it does not give worst case bound.

Worst-Case Bound After Clustering. The clustering-based model selection method also provides a worst-case bound. If each feature does not appear in more than l clusters, then the critical count of each feature is no more than lq because we only select q models per cluster. So the total critical count of k features will not exceed klq. The selected models satisfy $(klq, c*, k)$-robustness where $c*$ is the minimal sum of c_j of k features. For example, suppose in Fig. 1 the trees are divided into two clusters, the first cluster with tree f_1 and f_3 and the second with tree f_2. Each feature only shows up in one cluster, i.e., $l = 1$. We also select one model per cluster so $q = 1$. So the selected models (say f_1 and f_2) satisfy $(1, c*, 1)$-robustness and by Theorem 1 attackers need to modify at least 2 features for any instance with two positive votes.

Impact of Using Weighted Random Forest. We have two observations of the proposed weighted random forest method. First, suppose two features are identical except that one has higher cost than the other, WRF will favor the feature with higher cost. This is obvious from Equation (4) where higher the cost (c_j), higher the weight. Second, suppose two features A_i and A_j have similar information gain or differential ratio except that A_i is used in more trees generated by the original random forest method than A_j, WRF will favor A_j (the feature used in fewer trees). This is evident from Equation (3) where the feature appears in more trees will have higher total information gain (or differential ratio) and thus lower weight in Equation (4).

These observations mean that compared to original random forest algorithm, WRF will tend to pick features with higher cost or less frequently used in existing trees. Thus the pool of trees generated by WRF tends to have lower critical count bound (t_1) and higher L_0-distance bound (t_2) if number of features k is fixed. Lower t_1 means the pool is more likely to satisfy the necessary condition that number of positive votes need to be at least $\lceil M/2 \rceil + t_1$. Thus the worst case bound in Theorem 1 is more likely to hold. Higher t_2 means adversaries need to pay more effort in terms of L_0 distance. So WRF leads to more robust models in general.

6 Experimental Results

This section presents experimental evaluation of our proposed methods. Section 6.1 describes setup of our experiment. Section 6.2 discusses how we tune parameters of proposed methods and Section 6.3 compares proposed methods to existing ones.

6.1 Setup

Algorithms. We compare the following algorithms:

1) WRFD: this is the proposed weighted random forest algorithm using differential ratio in the weighting scheme. Clustering is not used.
2) WRFI: this is WRF using information gain in the weighting scheme.
3) CWRF: this is the proposed algorithm with both weighting at model building time and cluster-based model selection (in Algorithm 3) at model application time. We also use WRFD to create the model pool because WRFD has better results than WRFI in most cases.
4) RF: this is the original random forest algorithm. RF can be seen as a special case of WRFD or WRFI when $r = 0$ (i.e., weights are uniform).
5) Retraining-Tree: this method was used in [31]. It takes all true positive training instances (i.e., those positive instances also classified as positive by the current model) and finds for each training instance x_i an adversarial instance x_i' to evade detection using

TABLE 2
Characteristics of Data Sets

Data set	Number of instances	Number of features	Number of positive instances
Spambase	4,601	57	1813
Kyoto-Sample	45,390	14	22,687

TABLE 3
Groups and Costs for Features in Kyoto-Sample

Groups	Features	Costs
Group 1	Duration, SourceBytes, Source_Port_Number	1
Group 2	Count, Same_srv_rate, Serror_rate, Srv_Serror_Rate, Dst_Host_Count, Dst_Host_Srv_Count, Dst_Host_Same_Src_Port_Rate, Dst_Host_Serror_Rate	3
Group 3	DestinationBytes, Destination_Port_Number	5

a Greedy algorithm which will be described later in this Section. Random forest models are then created using the augmented training set which contains both the original training instances and the adversarial instances. This process is also repeated (to have new models to classify training data and add adversarial instances from true positives) until no more adversarial instances can be generated.

6) Retraining-CNN: this method creates a convolutionary neural network (CNN) over the training data and then uses the JSMA method [50] to generate adversarial examples because JSMA minimizes L_0 distance. The adversarial examples are added to the original training data to retrain a more robust CNN.

Data Sets. We used the Spambase data set (an email spam data set) from UCI Machine Learning Repository [30] and network traffic data from Kyoto University's Honeypot (an intrusion detection data set) [51]. For the Kyoto University data set, we randomly selected a sample of 45,390 instances from data collected in December 2015. Since the original data set is quite skewed (with mostly normal traffic), we undersampled normal traffic data and kept about half sample normal and half attacks. We also removed duplicates from the data set. We call this sampled data set *Kyoto-Sample*. Details of these data sets can be found in Table 2. Spambase only contains numerical features. Kyoto University data has both numerical and categorical features.

Features for Spambase data are mainly frequency of words and symbols as well as length of sequence of capital letters. All these features are relatively easy to modify so we set c_j (the cost of modifying feature j in Equation (1)) for all features to one for Spambase data set.

For Kyoto University data set we only used features extracted from raw traffic data such as duration of connection, number of bytes sent by source IP. We divide these features into three groups as shown in Table 3 according to attackers' difficulty of modifying these features: for the feature like "Source Port", it is easy for attackers to modify by themselves, so we set c_j for these features to 1; for the features like "Destination Port", attackers can barely change the values in the real world, we set c_j for these features to 5; for the rest of features like "Count" or "Serror_Rate", it may need attackers to take some efforts to modify but not too much, so we set c_j of these features to 3.

For Spambase, we randomly selected 70 percent of data for training and the remaining for testing. For Kyoto-Sample, we randomly selected 10 days of data for training and the remaining for testing. We also assume that attackers do not have access to training data but do have access to testing data. Attackers will try to evade detection for every true positive instance in the testing data. Note that there is no need to modify false negative instances as they already evade detection.

Attacking Strategies. The error function for random forest is not differentiable. So we cannot use methods such as gradient descent. Instead we used a greedy attacking method to find adversarial instances (a similar method is used in [31]). This method tries to find adversarial instances closest to x_i based on L_0 distance.

Given a true positive instance x_i in the testing data, the Greedy method iteratively selects a feature to modify such that this will lead to maximal reduction of the count of positive votes divided by weight c_i. Here count of positive votes is the number of trees that classify the current modified instance x_i' as positive. Clearly, if positive vote count is less than half of total number of trees then x_i' will be classified as negative and become an adversarial instance.

In the white-box case, the Greedy method first finds a negative instance x_n that has the lowest positive vote count. So x_n looks the most negative according to the detection model. It then checks every unmodified feature and replaces that feature with the value of the feature in x_n (other features remain unchanged) and computes the new positive vote count. For Spambase data set, the cost of modifying each feature (c_j) is one, so the feature A_j^* with the lowest positive vote count will be selected. For Kyoto-Sample data set, attackers will consider different difficulty of modifying features when they select features using greedy method. We assume attackers will divide vote change of each feature by its predefined cost c_j and select feature A_j^* with the highest result. This process is repeated until majority of trees classify the modified instance x_i' as negative.

In the black-box case, attackers do not know the tree models built from training data. The attackers randomly select a subset of testing instances they have and learn a set of tree models based on this subset. Let's call this model pool $\mathcal{P}'$. The greedy method is then applied just as the white-box case using $\mathcal{P}'$ to help it choose the feature to modify. At the end of each iteration when a feature A_j^* is selected, the Greedy method will probe the original models to check whether the original model pool $\mathcal{P}$ will classify the modified instance x_i' (with feature A_j^* modified) as negative and stops if this happens. In both data sets we used half of the testing data for attackers to build their own model $\mathcal{P}'$.

Metrics. One way to measure robustness is to run all models on real evasion attack data. However it is quite difficult to find such data sets. So in the literature robustness is often measured by L_p distance between adversarial instances and the original instances.

We also assume that attackers can modify as many features as they want and as much per feature as they want.

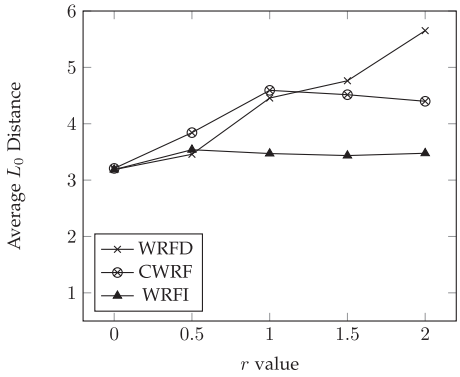


Fig. 4. Average L_0 distance when varying r on Spambase (white-box).

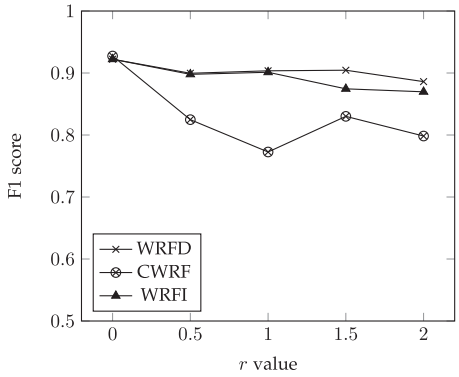


Fig. 6. F1 score when varying r on Spambase.

We used average L_0 distance between adversarial instances and their corresponding original instances. For Spambase data set, we set all c_j to one, so the average L_0 distance per instance is the average number of features modified by the attackers to evade each attack instance. For Kyoto-Sample we assigned non uniform weights listed in Table 3. In both cases, the higher the average distance, the more robust the model.

For Kyoto-Sample data set, since cost of modifying a feature is not uniform, it is a little difficult to interpret L_0 distance. To address this issue, we also consider a case when attackers can only launch attacks with the total cost not exceeding a budget b. We measured success rate (fraction of attacks that can evade detection) of such adversarial attacks for such bounded attack case.

We also measure a model's ability to detect original non-evasive attacks using precision, recall, and F1 score over original testing data which only contains original attacks and normal data.

All experiments were run on a desktop computer with Intel $i7$ quad core processor, 32 GB RAM, 2 TB hard disk, and running Windows 10. All algorithms were implemented in Java by extending source code for Weka 3.8. Since there is some randomness in mining algorithms and attacking strategies, we ran each experiment 20 times and took the average of results.

6.2 Tuning of Parameters

We need to set three parameters for our proposed WRFD, WRFI and CWRF methods: 1) r which is used in Equation (4); 2) s as the number of clusters for CWRF; 3) q as the number of models selected at model application time from each cluster of models. All three parameters can be used to adjust the tradeoff between defense against evasion attacks (robustness) and defense against non-evasive (original) attacks.

Tuning of r. we found that results for white-box and black-box cases have similar trends so we only report results for white-box. As mentioned in Section 6.1, average L_0 distance is used to measure robustness to evasion attacks. Figs. 4 and 5 report average L_0 distance for each data set under white-box attack. Figs. 6 and 7 report F1 score for Spambase and Kyoto-Sample, respectively. We reported results for WRFD, CWRF, and WRFI because all these methods use r. For CWRF we used $s = 10, q = 3$ for Spambase and $s = 5, q = 1$ for Kyoto-Sample.

The results showed that as r increases, the average L_0 distance (robustness) increases and F1 score decreases. This is expected because higher r values put more emphasis on robustness. So there is a need to select r to balance defense against evasion attacks and non evasive (original) attacks.

The increase in robustness also becomes flat for large r values in some cases (e.g., for WRFI and CWRF on Spambase). In such cases, the weights for vulnerable features are so low such that other features that are previously not vulnerable may have higher weights and become vulnerable (i.e., likely to appear in many trees).

For Spambase, the F1 score of WRFD and WRFI only drops slightly as r increases. The F1 score of CWRF is more sensitive to r. We set r to 1.5 for Spambase as it achieves the best tradeoff between defense against original attacks and evasion attacks. For Kyoto-Sample, the F1 score of all methods are quite high and are not very sensitive to r. So we

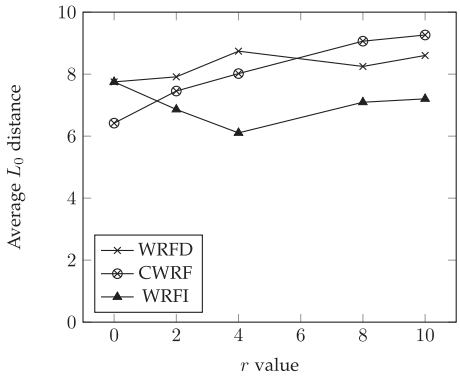


Fig. 5. Average L_0 distance when varying r on Kyoto-Sample (white-box).

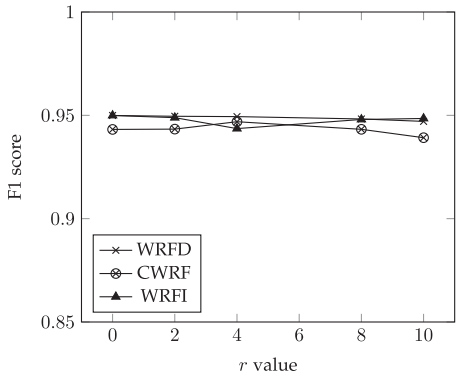


Fig. 7. F1 score when varying r on Kyoto-Sample.

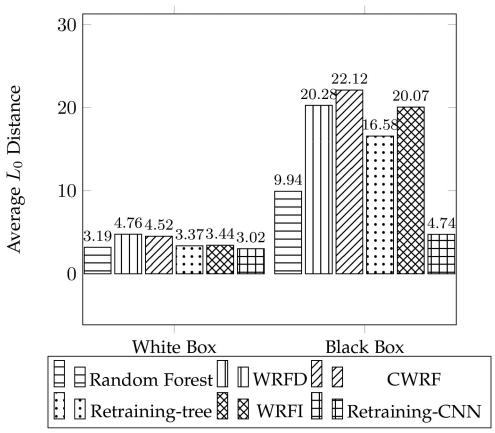


Fig. 8. Average L_0 distance on Spambase.

select $r = 10$ for Kyoto-Sample because that will maximize robustness. In practice, users should start with small r values and gradually increase it until L_0 distance or F1 score for original attacks starts to drop significantly.

In subsequent experiments, r is set to 1.5 on Spambase and 10 on Kyoto-Sample.

Tuning of s and q. we considered four combinations of s and q: $s = 5$ or 10 (i.e., models are divided into 5 or 10 clusters) and $q = 1$ or 3 (1 or 3 models are selected per cluster). Again we found the optimal value for s and q is the same for both black-box and white-box cases. Due to lack of space we just summarize the results.

We found that as s and q increases, F1 score increases because more models are used in prediction. However the average L_0 distance decreases in most cases as s and q increase because using more models means less uncertainty and less robustness. For Spambase, the decrease in F1 score is quite significant as s and q decrease so the optimal setting is using 10 clusters and selecting 3 models per cluster. For Kyoto-Sample data set, F1 score is not very sensitive to s and q. So we use 5 clusters and select 1 model from each cluster ($s = 5, q = 1$) to achieve better robustness.

6.3 Comparison With Existing Methods

Once we set parameters for our algorithms (WRFD, WRFI and CWRF), we compare them to other methods (RF and Retraining-tree). Figs. 8 and 9 report average L_0 distance for

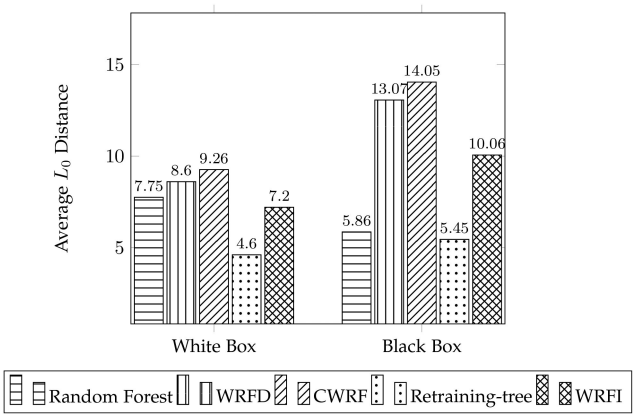


Fig. 9. Average L_0 distance on Kyoto-Sample.

TABLE 4
Precision, Recall, and F1 Score on Spambase

Algorithms	Precision	Recall	F_1 Score
Random Forest	0.932	0.912	0.922
WRFD	0.905	0.905	0.905
CWRF	0.81	0.849	0.829
Retraining-tree	0.871	0.963	0.914
WRFI	0.911	0.841	0.874
Retraining-CNN	0.870	0.960	0.913

Spambase and Kyoto-Sample, respectively. Tables 4 and 5 report precision, recall, and F1 score for Spambase and Kyoto-Sample, respectively. The Retraining-CNN method is not run on Kyoto-Sample because this data set has 14 features selected by domain experts, while the main benefit of deep learning is feature engineering which is not needed in this case.

All methods are more robust under black-box attacks than under white-box attacks. This is expected as under black-box attacks attackers have no knowledge of the original models.

Results on Spambase. Since $c_i = 1$ for Spambase, average L_0 distance is equivalent to average number of features attackers need to modify to evade detection.

On Spambase, WRFD and CWRF have the highest robustness (measured by average L_0 distance) under white-box attacks. CWRF also beats WRFD under black-box attacks due to increased uncertainty at model application time. WRFI has lower L_0 distance than WRFD under white-box attacks, showing that information gain is not the most appropriate measure for vulnerability in this case.

Retraining-tree surprisingly has very low L_0 distance under white-box attacks. We checked the trees created by this method. We found that four features still appear in over 80 percent of trees. Retraining-tree method still use random forest on the augmented training set and it still uses a few features in most trees. These features are still vulnerable to evasion attacks. Under black-box case Retraining-tree has better L_0 distance possibly due to the increased difficulty for attackers to find vulnerable features, but it is still less robust than CWRF, WRFD and WRFI.

The Retraing-CNN method has a L_0 distance of 3.02 under white-box attacks. Its L_0 distance is 4.74 under black-box attacks. These numbers are better than the original CNN model which has a L_0 distance around 1.6, but are worse than the results of our proposed methods CWRF and WRFI.

The improvement of CWRF and WRFD over RF is also quite significant. Under white-box attack the attackers need to modify on average 4.76 features for WRFD and 4.5 features for CWRF versus 3.19 features for RF. Under black-box attack

TABLE 5
Precision, Recall, and F1 Score on Kyoto-Sample

Algorithms	Precision	Recall	F_1 Score
Random Forest	0.944	0.956	0.950
WRFD	0.941	0.954	0.947
CWRF	0.931	0.948	0.940
Retraining-tree	0.906	0.957	0.931
WRFI	0.941	0.956	0.948

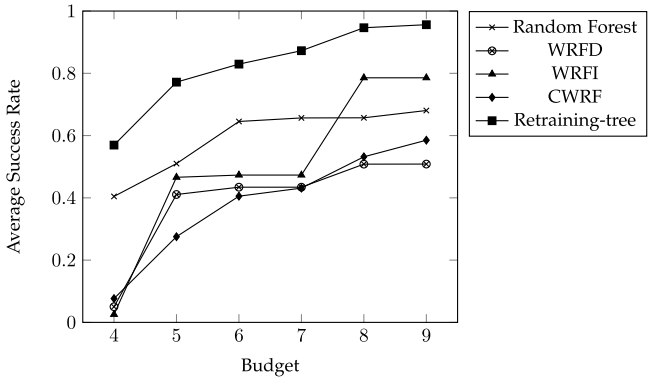


Fig. 10. Average success rate of white-box attacks when varying budget on Kyoto-Sample.

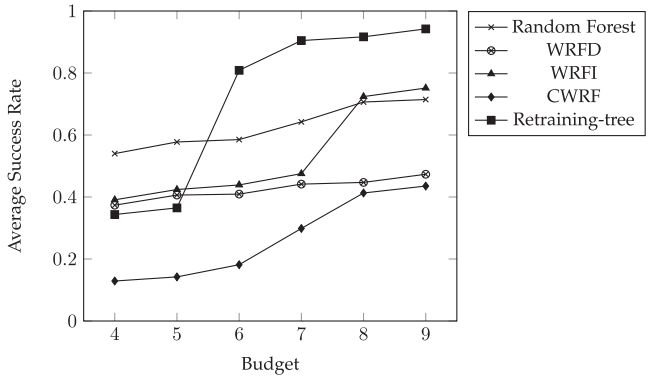


Fig. 11. Average success rate of black-box attacks when varying budget on Kyoto-Sample.

the attackers need to modify 20 out of 57 features for WRFD versus 9.9 features for RF.

In terms of detection of original attacks, the proposed methods have slightly lower prevision, recall, and F1 score than the original RF. WRFD has the best performance among proposed methods, with just 1-2 percent degrade in precision, recall and F1 score.

Retraining-tree and Retraining-CNN have similar precision, recall, and F1 score with those of RF. However as mentioned above both have limited improvement in terms of robustness compared to RF.

Overall, WRFD is the most effective against both original non-evasive attacks and evasion attacks on this data set as its precision, recall, and F1 score for original attacks are quite close to those of RF and its robustness is the highest (under white-box attacks) or the second highest (under black-box attacks).

Results on Kyoto-Sample. For Kyoto-Sample, all methods have very high precision, recall, and F1 score (over 91 percent) for original attacks. The difference between different methods is quite small. So we focus on robustness.

CWRF has the highest L_0 distance under both white-box and black-box attacks. The gap between CWRF and the other methods is quite significant. E.g., under black-box attacks the average L_0 distance is 14.05 for attackers to evade detection for CWRF models versus an average L_0 distance of 5.86 for RF models. CWRF is the best method for this data set.

On this data set WRFD has the second highest average L_0 distance, followed by WRFI. Retraining-tree still has the lowest average L_0 distance compared to WRFD, CWRF, and WRFI. Since there are fewer features Retraining-tree method still uses a few features in most trees, making them vulnerable to evasion attacks.

Statistical Significance. We also ran t-test to chech whether the differences of L_0 distance between the proposed algorithms (CWRF, WRFD, and WRFI) and existing ones (RF and Retraining-tree) are statistical significance. All the p values are smaller than 0.001 so the results are statistically significant. For instance, the p value of RF and WRFD for black-box attack in Spambase data is 5.40068E-50, and the p value of Retraining-tree method and CWRF for white-box attack in Kyoto-Sample data is 7.1744E-145. The complete results are not listed due to space limit.

Attacks on Kyoto-Sample With a Bounded Budget. Since L_0 distance does not directly translate to number of features for Kyoto data due to non uniform weights of different features, we also reported success rate of both white-box and black-box attacks when L_0 distance is bounded by a value b. We call b the *budget* of attacks and attackers can only generate attack instances with L_0 distance less or equal to b. Fig. 10 shows the white-box attacks' success rate when attackers' budget b was varied from 4 to 9. Fig. 11 shows the results for black-box attacks.

When attackers' budget increases, success rate increases for all methods. This is expected as attackers can modify more features.

Our proposed methods lead to significantly lower success rate than existing methods (RF and Retraining-tree). For example, when the budget is 4, success rate of white-box attacks for proposed CWRF is 0.074 while the success rates for RF and Retraining-tree are 0.4 and 0.57, respectively.

Among all the methods, the proposed CWRF method achieves the lowest success rate in most cases. Another proposed method WRFD is second the best in most cases and its results are close or better than results of CWRF when budget reaches 9. This suggests clustering and random selection of a subset of models work better when attackers have limited budget probably because in such cases attackers can only pick very few features to modify and random selection of models will make it very difficult for attackers to pick the right features. WRFI in general has worse results than WRFD and CWRF because it uses information gain. RF and Retraining-tree both have very high success rates, indicating these methods are not robust to evasion attacks.

Scalability. We extracted 20, 40, 60, 80 percent as well as the full set of Kyoto data set collected in December 2015 (with 7,565,246 million instances in total) to test the scalability of CWRF. Fig. 12 reports the execution time of CWRF, RF, and Retraining-Tree. The results show that CWRF scales linearly with the number of rows. Its execution time is slightly more than twice the execution time of RF, which is expected as CWRF needs to build the model pool twice and needs to compute weights of each feature. However, CWRF is more efficient than Retraining-Tree because the retraining method not only needs to train the model twice, but also needs to generate adversarial examples. We also found that the execution time of CWRF is dominated by model building time (clustering time is less than 10 seconds). As future work we will investigate how to further improve efficiency of our methods.

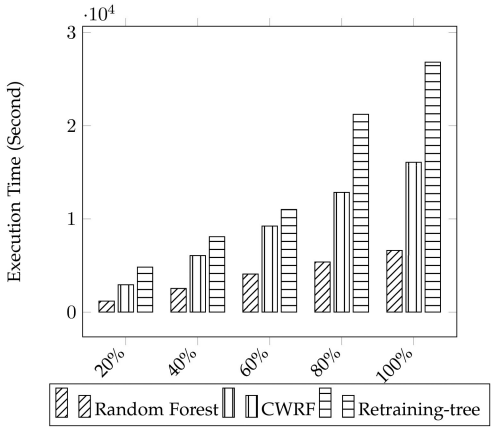


Fig. 12. Execution time of CWRF, Random Forest, and Retraining-tree over fraction of Kyoto December 2015 data.

7 Discussion of Extending Our Approach to Other Mining Methods

Now we discuss how to extend our approach to other mining methods. We can divide other mining methods into two categories. The first category requires well defined features and most mining methods other than deep neural nets belong to this category. The second category (deep neural nets) does not have well defined features and features are generated and selected in the learning process. We will consider how to extend our approach to the second category in future. Here we discuss how to extend our approach to the first category of mining methods.

The extended method works in two steps. In the first step, a pool of mining models are trained using bagging and random subspace. More specifically, each time we draw a random sample of training data and select a random subset of features and build a model using the sample and selected subset of features. Note that this step is similar to random forest which also uses bagging and random subspace.

The training method also should take into account vulnerability of each feature such that more vulnerable features receive lower weights. One possible way is to compute a vulnerability score v_j for feature A_j.

To compute v_j, we can further divide mining methods into two subcategories: those whose prediction function can be represented as a simple function and those cannot. For example, linear regression, logistic regression, and SVM belong to the first subcategory, and decision trees, Naive Bayesian, and Neural Network (not deep ones) belong to the second subcategory. For the first subcategory, the prediction of an instance z can be represented as a function $f(z)$. So we can simply take the partial derivative of f over z_j (the jth feature of z). The partial derivative represents how much the prediction will change given a change of z_j. We can set v_j to this partial derivative because a feature with the largest partial derivative has the most impact on prediction. Note that a similar idea has been used in [24].

For the second subcategory, we can use some heuristics to compute weights as what we did for random forest. For example, we can use a similar idea as differential ratio for Naive Bayesian. If a feature A_j's value v is very common in positive class, but very rare in negative class, then modifying v is likely to avoid detection. So we can define differential ratio $d(v)$ as $P(v|c_+) - P(v|c_-)$ where c_+ is the positive class and c_- is negative class. A_j's differential ratio is the maximal ratio of all its values and the vulnerability measure v_j of A_j in a pool of models is the sum of its differential ratio in all models.

Once v_j is computed, it can be used in the model building algorithm to put more emphasize on less vulnerable features. For most methods, this can be done by integrating the weights in objective function. For example, for SVM, suppose RBF kernel is used and the kernel function is $K(x_i, z) = e^{-\gamma \sum_{j=1}^{m} (x_{ij} - z_j)^2}$. We can replace this function with a weighted kernel function $K'(x_i, z) = e^{-\gamma \sum_{j=1}^{m} w_j (x_{ij} - z_j)^2}$. Since $w_j = e^{-v_j/c_j}$, this new kernel function favors features with low vulnerability (v_j) and high modification cost.

In the second step, a similar cluster-based method is used to select a subset of models at model application time. The generated models are divided into s clusters using spectral clustering. Finally at model application time for each test case z, the method randomly selects q models from each cluster and uses the qs selected models to classify z.

8 Conclusion

This paper proposes an approach to use randomization to improve robustness of tree based models in cyber security applications. Our approach injects randomness into both model training time and model application time and is effective against both evasion attacks and original (non-evasive) attacks. We applied our approach to random forest and experiments on an email spam data set and a network intrusion detection data set show our approach significantly improves robustness of random forest models without sacrificing much effectiveness for detecting original attacks. The proposed methods also beat retraining method, indicating that making the learning algorithm aware of robustness is more effective than just adding adversarial examples to the training data.

We also proposed a theoretical framework to provide worst case bound on attackers' efforts and discussed possible ways to extend our approach to other mining methods. As future work we will investigate how to implement these extensions.

Acknowledgments

This work was partially supported by Office of Naval Research grant N00014-18-1-2452.

References

[1] E. Blanzieri and A. Bryl, "A survey of learning-based techniques of Email spam filtering," *Artif. Intell. Rev.*, vol. 29, no. 1, pp. 63–92, 2008.

[2] G. V. Cormack, "Email spam filtering: A systematic review," *Found. Trends Inf. Retrieval*, vol. 1, no. 4, pp. 335–455, 2007.

[3] M. Siddiqui, M. C. Wang, and J. Lee, "A survey of data mining techniques for malware detection using file features," in *Proc. 46th Annu. Southeast Regional Conf. XX*, 2008, pp. 509–510.

[4] W. Lee, S. J. Stolfo, and K. W. Mok, "A data mining framework for building intrusion detection models," in *Proc. IEEE Symp. Security Privacy*, 1999, pp. 120–132.

[5] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, 2009, Art. no. 15.

[6] S. Axelsson, "Intrusion detection systems: A survey and taxonomy," Chalmers University of Technology, Goteborg, Sweden, Tech. Rep. 99-15, 2000.

[7] L. Portnoy, E. Eskin, and S. Stolfo, "Intrusion detection with unlabeled data using clustering," in *Proc. ACM CSS Workshop Data Mining Appl. Secur.*, 2001, pp. 5–8.

[8] D. Lowd and C. Meek, "Adversarial learning," in *Proc. 11th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2005, pp. 641–647.

[9] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," *IEEE Access*, vol. 6, pp. 14410–14430, 2018.

[10] I. Evtimov *et al.*, "Robust physical-world attacks on machine learning models," *CoRR*, 2017. [Online]. Available: http://arxiv.org/abs/1707.08945

[11] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Proc. IEEE Eur. Symp. Security Privacy*, 2016, pp. 372–387.

[12] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial perturbations against deep neural networks for malware classification," 2016, *arXiv:1606.04435*.

[13] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter, "Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 1528–1540. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978392

[14] T. Gu, B. Dolan-Gavitt, and S. Garg, "BadNets: Identifying vulnerabilities in the machine learning model supply chain," 2017, *arXiv:1708.06733*.

[15] W. Liu and S. Chawla, "A game theoretical model for adversarial learning," in *Proc. IEEE Int. Conf. Data Mining Workshops*, 2009, pp. 25–30.

[16] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, "Can machine learning be secure?" in *Proc. ACM Symp. Inf. Comput. Commun. Secur.*, 2006, pp. 16–25.

[17] M. Brückner, C. Kanzow, and T. Scheffer, "Static prediction games for adversarial learning problems," *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 2617–2654, 2012.

[18] M. Kantarcioglu, B. Xi, and C. Clifton, "A game theoretical framework for adversarial learning," in *Proc. CERIAS 9th Annu. Inf. Secur. Symp.*, 2008, p. 1, Art. No. 26.

[19] M. Kantarcıoğlu, B. Xi, and C. Clifton, "Classifier evaluation and attribute selection against active adversaries," *Data Mining Knowl. Discov.*, vol. 22, no. 1/2, pp. 291–335, 2011.

[20] M. Ramoni and P. Sebastiani, "Robust learning with missing data," *Mach. Learn.*, vol. 45, no. 2, pp. 147–170, 2001.

[21] A. Globerson and S. Roweis, "Nightmare at test time: Robust learning by feature deletion," in *Proc. 23rd Int. Conf. Mach. Learn.*, 2006, pp. 353–360.

[22] X.-T. Yuan and B.-G. Hu, "Robust feature extraction via information theoretic learning," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 1193–1200.

[23] A. Kołcz and C. H. Teo, "Feature weighting for improved classifier robustness," in *Proc. 6th Conf. Email Anti-Spam*, 2009.

[24] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. Int. Conf. Learn. Representations*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6572

[25] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Proc. IEEE Symp. Security Privacy*, 2016, pp. 582–597.

[26] H. Lee, S. Han, and J. Lee, "Generative adversarial trainer: Defense to adversarial perturbations with GAN," 2017, *arXiv:1705.03387*.

[27] P. D. Grünwald, *The Minimum Description Length Principle*. Cambridge, MA, USA: MIT Press, 2007.

[28] B. Biggio, G. Fumera, and F. Roli, "Adversarial pattern classification using multiple classifiers and randomisation," in *Proc. Joint IAPR Int. Workshops Structural Syntactic Statist. Pattern Recognit.*, 2008, pp. 500–509.

[29] B. Biggio, G. Fumera, and F. Roli, "Multiple classifier systems for robust classifier design in adversarial environments," *Int. J. Mach. Learn. Cybern.*, vol. 1, no. 1/4, pp. 27–41, 2010.

[30] S. Hettich, C. Blake, and C. Merz, "UCI repository of machine learning databases," 1998. [Online]. Available: http://www.ics.uci.edu/ mlearn/MLRepositoryhtml

[31] A. Kantchelian, J. Tygar, and A. Joseph, "Evasion and hardening of tree ensemble classifiers," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 2387–2396.

[32] B. Biggio *et al.*, "Evasion attacks against machine learning at test time," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discov. Databases*, 2013, pp. 387–402.

[33] P. Fogla, M. I. Sharif, R. Perdisci, O. M. Kolesnikov, and W. Lee, "Polymorphic blending attacks," in *Proc. 15th Conf. USENIX Secur. Symp.*, 2006, Art. no. 17.

[34] B. Nelson *et al.*, "Query strategies for evading convex-inducing classifiers," *J. Mach. Learn. Res.*, vol. 13, no. May, pp. 1293–1332, 2012.

[35] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs," in *Proc. 25th USENIX Conf. Secur. Symp.*, 2016, pp. 601–618.

[36] O. Dekel, O. Shamir, and L. Xiao, "Learning to classify with missing and corrupted features," *Mach. Learn.*, vol. 81, no. 2, pp. 149–178, 2010.

[37] G. R. Lanckriet, L. E. Ghaoui, C. Bhattacharyya, and M. I. Jordan, "A robust minimax approach to classification," *The J. Mach. Learn. Res.*, vol. 3, pp. 555–582, 2003.

[38] C. H. Teo, A. Globerson, S. T. Roweis, and A. J. Smola, "Convex learning with invariances," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2007, pp. 1489–1496.

[39] Y. Zhou, M. Kantarcioglu, B. Thuraisingham, and B. Xi, "Adversarial support vector machine learning," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2012, pp. 1059–1067.

[40] M. Kearns and M. Li, "Learning in the presence of malicious errors," *SIAM J. Comput.*, vol. 22, pp. 807–837, 1993.

[41] N. Dalvi, P. Dominggos, S. S. Mausam, and D. Verma, "Adversarial classification," in *Proc. 10th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2004, pp. 99–108.

[42] M. Brückner and T. Scheffer, "Stackelberg games for adversarial prediction problems," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2011, pp. 547–555.

[43] W. Liu and S. Chawla, "Mining adversarial patterns via regularized loss minimization," *Mach. Learn.*, vol. 81, no. 1, pp. 69–83, 2010.

[44] A. A. Ross, K. Nandakumar, and A. K. Jain, *Handbook of Multibiometrics*, vol. 6. Berlin, Germany: Springer, 2006.

[45] S. Hershkop and S. J. Stolfo, "Combining Email models for false positive reduction," in *Proc. 11th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2005, pp. 98–107.

[46] T. P. Tran, P. Tsai, and T. Jan, "An adjustable combination of linear regression and modified probabilistic neural network for anti-spam filtering," in *Proc. 19th Int. Conf. Pattern Recognit.*, 2008, pp. 1–4.

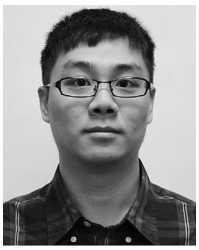
[47] R. Perdisci, G. Gu, and W. Lee, "Using an ensemble of one-class SVM classifiers to harden payload-based anomaly detection systems," in *Proc. 6th Int. Conf. Data Mining*, 2006, pp. 488–498.

[48] D. B. Skillicorn, "Adversarial knowledge discovery," *IEEE Intell. Syst.*, vol. 24, no. 6, pp. 0054–61, Nov./Dec. 2009.

[49] Y. Vorobeychik and B. Li, "Optimal randomized classification in adversarial settings," in *Proc. Int. Conf. Auton. Agents Multi-Agent Syst.*, 2014, pp. 485–492.

[50] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Proc. IEEE Eur. Symp. Secur. Privacy*, 2016, pp. 372–387.

[51] J. Song, H. Takakura, Y. Okabe, M. Eto, D. Inoue, and K. Nakao, "Statistical analysis of honeypot data and building of kyoto 2006+ dataset for nids evaluation," in *Proc. 1st Workshop Building Anal. Datasets Gathering Experience Returns Secur.*, 2011, pp. 29–36.



Fan Yang received the BS degree in software engineering from Hebei University, China, and the MS degree in computer science from the University of California, Santa Cruz, Santa Cruz, California. He is currently working toward the PhD degree in Information Systems Department, University of Maryland Baltimore County, Baltimore, Maryland. His research area is algorithm for privacy preserving data mining and adversarial learning.

Zhiyuan Chen received the BS and MS degrees from Fudan University, China, and the PhD degree in computer science from Cornell University, Ithaca, New York. He is currently an associate professor with the Department of Information Systems, University of Maryland Baltimore County, Baltimore, Maryland. His research covers the areas of data science, big data, privacy preserving data mining and data management, data exploration and navigation, semantic-based search and data integration using semantic networks, and adversarial learning and its applications in cyber security. He has published extensively in these areas and has received funding from NSF, IBM, Office of Naval Research, MITRE, and the Department of Education. He is managing editor of one journal and serves on editorial board of three other journals and was guest editor of a special issue.

Aryya Gangopadhyay received the PhD degree in computer information systems from Rutgers University, Camden, New Jersey. He is a professor of the Department of Information Systems, University of Maryland Baltimore County (UMBC), Baltimore, Maryland. He has been a faculty member at UMBC since 1997. He has mentored and graduated 16 PhD students who are working either as faculty members in various universities or holding leading IT positions in the private industries and government sectors. He has published five books and more than 125 peer-reviewed research articles. His research interests are in the area of data science and machine learning. His current research includes machine learning-based solutions in areas such as cybersecurity, multi-modal data fusion for emergency response, and healthcare applications such as computational drug repurposing. His research has been funded by grants from NSF, NIST, US Department of Education, IBM, Maryland Department of Transportation, and other agencies. For more information,please visit https://sites.google.com/site/homearyya/.