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

Fan Yang
University of Maryland, Baltimore County
Baltimore, MD
fyang1@umbc.edu

Zhiyuan Chen
University of Maryland, Baltimore County
Baltimore, MD
zhchen@umbc.edu

Aryya Gangopadhyay
University of Maryland, Baltimore County
Baltimore, MD
gangopad@umbc.edu

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 underexplored. 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. Experiments on intrusion detection and spam filtering data show that our approach further improves robustness of random-forest method.

CCS CONCEPTS

• **Security and privacy → Intrusion/anomaly detection and malware mitigation**; • **Computing methodologies → Machine learning**.

KEYWORDS

Evasion attacks, Machine Learning, Adversarial Learning, Intrusion Detection, Spam Filtering

ACM Reference Format:
Fan Yang, Zhiyuan Chen, and Aryya Gangopadhyay. 2019. Using Randomness to Improve Robustness of Tree-based Models Against Evasion Attacks. In *Fifth ACM International Workshop on Security and Privacy Analytics (IWSPA '19), March 27, 2019, Richardson, TX, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3309182.3309186

*Produces the permission block, and copyright information

IWSPA '19, March 27, 2019, Richardson, TX, USA

ACM ISBN 978-1-4503-6178-1/19/03…$15.00
https://doi.org/10.1145/3309182.3309186

1 INTRODUCTION

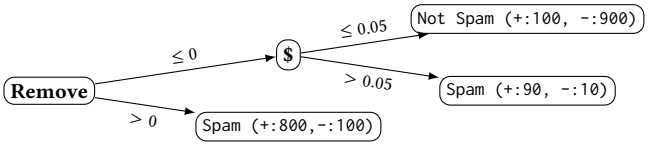
With the arrival of big data era, machine learning techniques have been widely used to build models for cyber security applications such as spam filtering [7, 11], malware or virus detection [42], and intrusion detection [2, 10, 30, 38]. However, attackers may use a type of attacking strategy called *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 [33], especially more recently on deep learning models [1, 14, 18, 20, 35, 41]. There also has been work on building more robust mining models against evasion attacks [3, 8, 16, 17, 23, 24, 27, 29, 31, 33, 36, 39, 49]. 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.

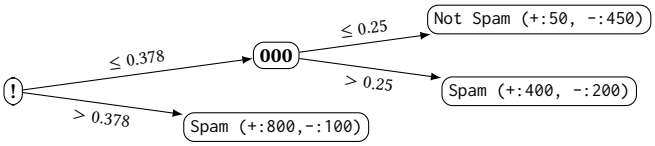
In addition, most existing studies use deterministic models. However for tree-based models, a deterministic model should be concise to avoid overfitting the training data according to the Minimum Description Length (MDL) principle [19]. A concise model often uses a small number of features or a small 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. Figure 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 in an email that contains a word or symbol 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 instances in each node.

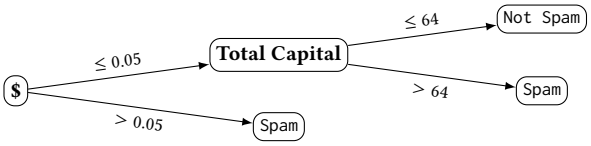
If only one decision tree, say f_1 is used, it is very easy for attackers to modify a spam email to evade detection. For example, suppose attackers have an email with feature values shown in Table 1(a). Attackers may learn that the spam filtering software looks at the "remove" and "$" features. As a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



(a) Model f_1



(b) Model f_2



(c) Model f_3

Figure 1: Decision tree models created by our model in 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

Table 1: Some possible attacks for Example 1

Some researchers try to use an ensemble approach (i.e., build a number of models instead of one) [5, 6] 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.

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 *feature bagging*). However we found random forest often fails to generate robust models. For example, We ran random forest on Spambase data set [22], which is an email spam data set with 57 features and built 100 decision trees. Figure 2 reports for each feature (attribute), the number of trees using that feature. 9 out of 57 features appear in at least 80% 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.

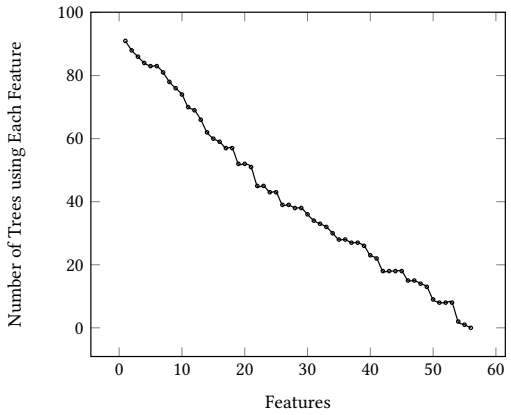


Figure 2: Number of trees using each feature by random forest on Spambase data set

Our contributions: This paper proposes a weighted random forest approach to generate 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 Figure 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 Figure 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 random forest.
(2) Methods to randomly select a subset of models at model application time to further boost robustness.
(3) Experiments to compare our methods to existing methods. We also showed how to set parameters in our methods to optimize the defense against both original 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 experimental results. Section 6 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 [4]. Barreno et al. described a taxonomy of attacks and briefly mentions a few possible defense strategies [3]. Lowd and Meek [33] 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 [15]. 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 [46].

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 [13, 28, 45]. Globerson and Roweis [16] 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 learning method is proposed for SVM in [50], 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 [26] proposed a theoretical upper bound on tolerable malicious error rates. Dalvi et al. [12] 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 [9, 24, 32] model the adversarial learning problem as a Stackelberg game.

The fourth category of work focuses on vulnerabilities of deep neural nets [14, 18, 20, 35, 41]. A survey can be found at [1]. 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 [17, 29, 36], 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 [18], 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 [21, 37, 40, 43, 47]. For example, Biggio et al. [6] 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 [25] 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 [48], 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 vs. 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 [3]).

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 made*

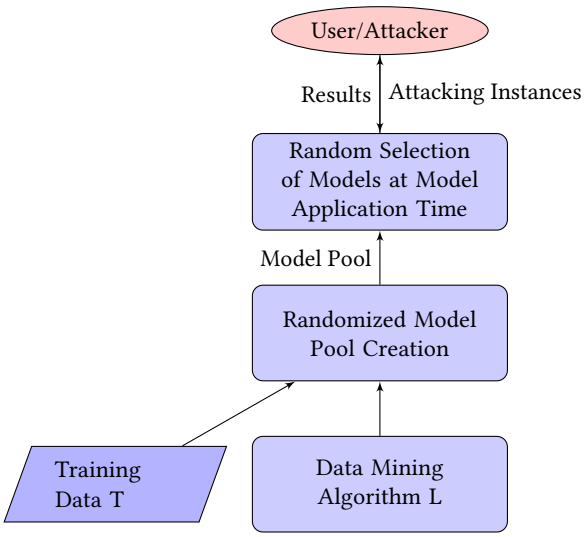


Figure 3: Architecture of Our Approach

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

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

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.

Figure 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: 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 $f_1, \ldots, f_M$

```
Compute weights w_1, ..., w_m for each feature
for i = 1 to M do
    Draw a uniform random sample with replacement of size
      N from T, let the sample be T_i
    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
end
return f_1, ..., f_M
```

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.

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 discuss how to quantify vulnerability of a feature.

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

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

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

We then define a feature A_j's information gain in a pool of trees as the sum of IG in each tree divided by M (total number of trees), where the IG in each tree is the maximal IG of all nodes having A_j as splitting attribute, i.e.,

$$IG(A_j, \mathcal{P}) = \frac{\sum_{f \in \mathcal{P}} \sum_{n \in f, n \text{ splits on } A_j} \max IG(A_j, n)}{M} \tag{3}$$

Here 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). Since the computation depends on a model pool $\mathcal{P}$, we can run the original random forest (not weighted one) once to create a model pool $\mathcal{P}_0$ and calculate the weight based on $\mathcal{P}_0$. We then compute weight w_j for feature A_j as

$$w(A_j) = e^{-r \frac{IG(A_j, \mathcal{P}_0)}{c_j}} \tag{4}$$

Here c_j is the difficulty of modifying A_j in Equation 1 and $\mathcal{P}_0$ is the pool built by random forest. 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 5.

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. We take the maximal ratio across all nodes splitting on A_j. We then take the average over all trees to get the differential ratio of $d(A_j, \mathcal{P}_0)$ over the model pool $\mathcal{P}_0$. We then replace $IG(A_j, \mathcal{P}_0)$ with $d(A_j, \mathcal{P}_0)$ in Equation 4 to compute the weight w_j.

Difference 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.

Figure 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

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

We then use this differential ratio in Equation 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 2). 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.

Algorithm 2: 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

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 5.

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).

5 EXPERIMENTAL RESULTS

This section presents experimental evaluation of our proposed methods. Section 5.1 describes setup of our experiment. Section 5.2 discusses how we tune parameters of proposed methods and Section 5.3 compares proposed methods to existing ones.

5.1 Setup

Algorithms: We compare the following algorithms:

(1) WRFD: this is the proposed weighted random forest algorithm (WRF) 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 2) 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: this method was used in [25]. 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 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.

Data sets: We used the Spambase data set (an email spam data set) from UCI Machine Learning Repository [22] and network traffic data from Kyoto University's Honeypot (an intrusion detection data set) [44]. 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 under-sampled 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. 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. Attackers can easily modify features in both data sets.

E.g., attackers can add or drop a word or symbol or capital letter sequences in a spam email for Spambase and modify the number of bytes sent by source IP for Kyoto data set.

For Spambase, we randomly selected 70% 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.

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 2: Characteristics of data sets

Attacking strategies: Given a malicious instance x_i, attackers want to find an adversarial instance x_i' that is closest to x_i. For simplicity we use L_0 distance and assume that all features have uniform c_j so attackers' effort will be minimized if they need to modify the smallest number of features.

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 [25]). This method tries to find adversarial instances closest to x_i based on L_0 distance (i.e., number of modified features).

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 (i.e., 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. The feature A_j^* with the lowest positive vote count will be selected. 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. We used average L_0 distance between adversarial instances and their corresponding original instances. Since we set all c_j to one, the average L_0 distance is the average number of features modified by the attackers to evade each attack instance. The higher the average, the more robust the model.

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

All experiments were run on a desktop computer with Intel *i*7 quad core processor, 32 GB RAM, 2 TB hard disk, and running Windows 7. 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.

5.2 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 5.1, average number of modified features is used to measure robustness to evasion attacks. Figure 4 and Figure 5 report average number of features modified by attackers for each data set under white-box attack. Figure 6 and Figure 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 number of features modified by attackers (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 on Kyoto-Sample). 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 select $r = 4$ for Kyoto-Sample because that will maximize robustness. In practice, users should start with small r values and gradually increase it until average number of

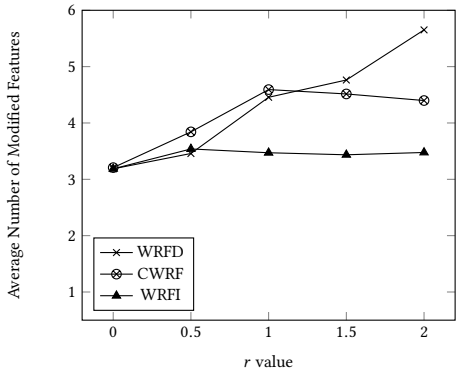


Figure 4: Average Number of Modified Features when varying r on Spambase (white-box)

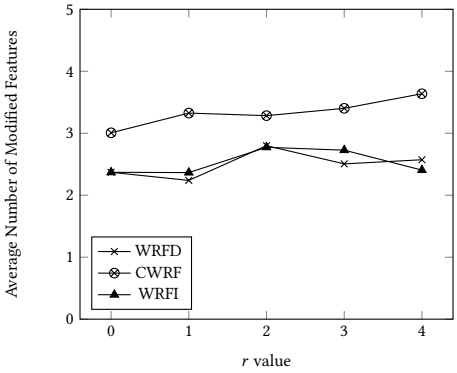


Figure 5: Average Number of Modified Features when varying r on Kyoto-Sample (white-box)

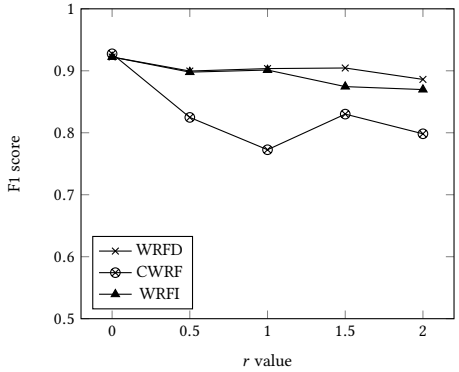


Figure 6: F1 score when varying r on Spambase

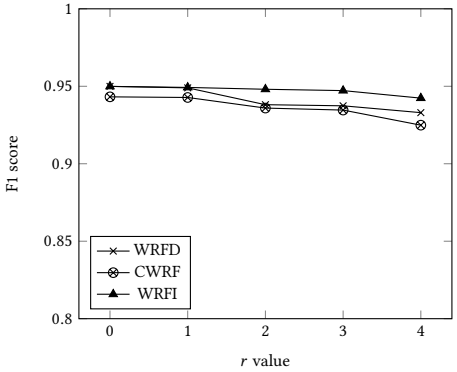


Figure 7: F1 score when varying r on Kyoto-Sample

modified features or F1 score for original attacks starts to drop significantly.

In subsequent experiments, r is set to 1.5 on Spambase and 4 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 number of modified features 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 ($s = 5, q = 1$).

5.3 Comparison with Existing Methods

Once we set parameters for our algorithms (WRFD, WRFI and CWRF), we compare them to other methods (RF and retraining). Figure 8 and Figure 9 report average number of modified features for Spambase and Kyoto-Sample, respectively. Table 3 and Table 4 report precision, recall, and F1 score for Spambase and Kyoto-Sample, respectively.

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

Retraining surprisingly has very low robustness under white-box attacks. We checked the trees created by this method. We found that four features still appear in over 80% of trees. Retraining 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 has better robustness possibly due to the increased difficulty for attackers to find vulnerable features, but it is still less robust than CWRF, WRFD and WRFI.

The least robust method is RF, which is expected. 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 vs. 3.19 features for RF. Under black-box attack the attackers need to modify 20 out of 57 features for WRFD vs. 9.9 features for RF.

In terms of detection of original attacks, WRFD and Retraining methods have slightly lower precision, recall, and F1 score than RF.

The difference between WRFD and RF is about 1-2%. CWRF and WRFI have lower prevision, recall, and F1 score than other methods. So 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%) for original attacks. The difference between different methods is quite small. So we focus on robustness.

CWRF has the highest robustness under both white-box and black-box attacks. The gap in terms of robustness between CWRF and the other methods is quite significant. E.g., under black-box attacks attackers need to modify on average 10.19 out of 14 features to evade detection for CWRF models vs. an average of 3.15 features for RF models. So CWRF is the best method for this data set.

On this data set WRFI and WRFD have similar robustness. Again retraining has very poor robustness. It is even more vulnerable than the original random forest method. Since there are fewer features retraining 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 number of modified features between the proposed algorithms (CWRF, WRFD, and WRFI) and existing ones (RF and retraining) 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 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.

Scalability: We extracted 20%, 40%, 60%, 80% 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. Figure 10 reports the execution time of CWRF. The results show that CWRF scales linearly with number of rows. We also found that the execution time 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.

6 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. We also made a number of simplified assumptions in our experiments (e.g., uniform cost of modifying each feature). In future work we will study results in more realistic settings, e.g., with different difficulty of modifying each feature and some features cannot be modified (e.g., the part of malware that carries out the attack). Using a real data set containing evasion attacks is also an interesting direction.

REFERENCES

[1] Naveed Akhtar and Ajmal Mian. 2018. Threat of adversarial attacks on deep learning in computer vision: A survey. *arXiv preprint arXiv:1801.00553* (2018).

[2] Stefan Axelsson. 2000. *Intrusion detection systems: A survey and taxonomy.* Technical Report. Technical report Chalmers University of Technology, Goteborg, Sweden.

[3] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D Joseph, and J Doug Tygar. 2006. Can machine learning be secure?. In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security.* ACM, 16–25.

[4] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases.* Springer, 387–402.

[5] Battista Biggio, Giorgio Fumera, and Fabio Roli. 2008. Adversarial pattern classification using multiple classifiers and randomisation. In *Structural, Syntactic, and Statistical Pattern Recognition.* Springer, 500–509.

[6] Battista Biggio, Giorgio Fumera, and Fabio Roli. 2010. Multiple classifier systems for robust classifier design in adversarial environments. *International Journal of Machine Learning and Cybernetics* 1, 1-4 (2010), 27–41.

[7] Enrico Blanzieri and Anton Bryl. 2008. A survey of learning-based techniques of email spam filtering. *Artificial Intelligence Review* 29, 1 (2008), 63–92.

[8] Michael Brückner, Christian Kanzow, and Tobias Scheffer. 2012. Static prediction games for adversarial learning problems. *the Journal of Machine Learning Research* 13, 1 (2012), 2617–2654.

[9] Michael Brückner and Tobias Scheffer. 2011. Stackelberg games for adversarial prediction problems. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 547–555.

[10] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41, 3 (2009), 15.

[11] Gordon V Cormack. 2007. Email spam filtering: A systematic review. *Foundations and Trends in Information Retrieval* 1, 4 (2007), 335–455.

[12] N. Dalvi, P. Dominggos, S. Sanghai Mausam, and D. Verma. 2004. Adversarial Classification. In *Proceedings of Tenth ACM SIGKDD Int. Con. on Knowledge Discovery and Data Mining.* ACM, 99–108.

[13] Ofer Dekel, Ohad Shamir, and Lin Xiao. 2010. Learning to classify with missing and corrupted features. *Machine learning* 81, 2 (2010), 149–178.

[14] Ivan Evtimov, Kevin Eykholt, Earlence Fernandes, Tadayoshi Kohno, Bo Li, Atul Prakash, Amir Rahmati, and Dawn Song. 2017. Robust Physical-World Attacks on Machine Learning Models. *CoRR* abs/1707.08945 (2017). arXiv:1707.08945 http://arxiv.org/abs/1707.08945

[15] Prahlad Fogla, Monirul I Sharif, Roberto Perdisci, Oleg M Kolesnikov, and Wenke Lee. 2006. Polymorphic Blending Attacks.. In *USENIX Security.*

[16] Amir Globerson and Sam Roweis. 2006. Nightmare at test time: robust learning by feature deletion. In *Proceedings of the 23rd international conference on Machine learning.* ACM, 353–360.

[17] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations.* http://arxiv.org/abs/1412.6572

[18] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. 2016. Adversarial perturbations against deep neural networks for malware classification. *arXiv preprint arXiv:1606.04435* (2016).

[19] Peter D Grünwald. 2007. *The minimum description length principle.* MIT press.

[20] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733* (2017).

[21] Shlomo Hershkop and Salvatore J Stolfo. 2005. Combining email models for false positive reduction. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining.* ACM, 98–107.

[22] S. Hettich, C.L. Blake, and C.J. Merz. 1998. UCI Repository of machine learning databases.

[23] Murat Kantarcioglu, Bowei Xi, and Chris Clifton. 2008. A Game Theoretical Framework for Adversarial Learning. In *CERIAS 9th Annual Information Security Symposium.* Citeseer.

[24] Murat Kantarcıoğlu, Bowei Xi, and Chris Clifton. 2011. Classifier evaluation and attribute selection against active adversaries. *Data Mining and Knowledge Discovery* 22, 1-2 (2011), 291–335.

[25] Alex Kantchelian, JD Tygar, and Anthony Joseph. 2016. Evasion and hardening of tree ensemble classifiers. In *International Conference on Machine Learning.* 2387–2396.

[26] M. Kearns and M. Li. 1993. Learning in the presence of malicious errors. *SIAM J. Comput.* 22 (1993), 807–837.

[27] Aleksander Kołcz and Choon Hui Teo. 2009. Feature weighting for improved classifier robustness. In *CEAS'09: sixth conference on email and anti-spam.*

[28] Gert RG Lanckriet, Laurent El Ghaoui, Chiranjib Bhattacharyya, and Michael I Jordan. 2003. A robust minimax approach to classification. *The Journal of Machine Learning Research* 3 (2003), 555–582.

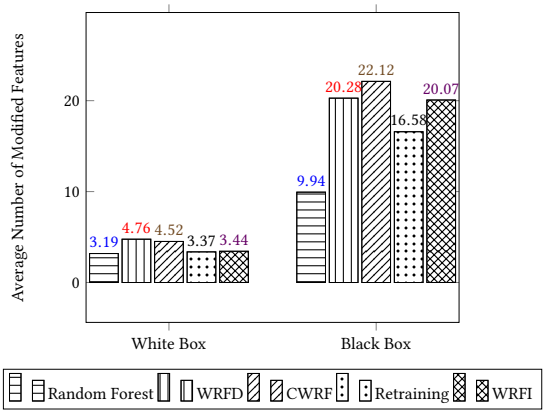


Figure 8: Average Number of Modified Features on Spambase

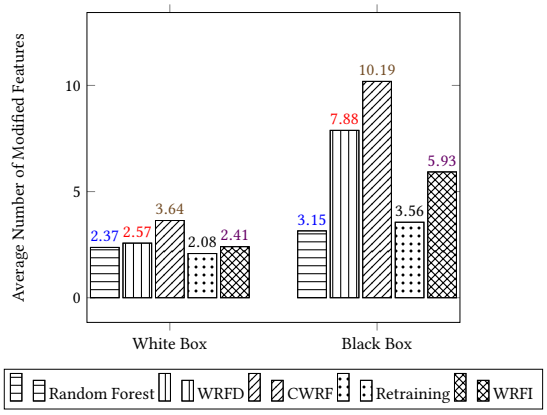


Figure 9: Average Number of Modified Features on Kyoto-Sample

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	0.871	0.963	0.914
WRFI	0.911	0.841	0.874

Table 3: Precision, Recall, and F1 score on Spambase

Algorithms	Precision	Recall	F_1 Score
Random Forest	0.944	0.956	0.950
WRFD	0.932	0.934	0.933
CWRF	0.922	0.928	0.925
Retraining	0.913	0.955	0.934
WRFI	0.933	0.952	0.942

Table 4: Precision, Recall, and F1 score on Kyoto-Sample

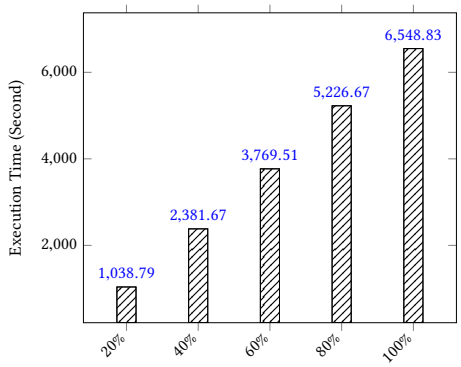


Figure 10: Execution time of CWRF over fraction of Kyoto December 2015 data

[29] Hyeungill Lee, Sungyeob Han, and Jungwoo Lee. 2017. Generative Adversarial Trainer: Defense to Adversarial Perturbations with GAN. *arXiv preprint arXiv:1705.03387* (2017).

[30] Wenke Lee, Salvatore J Stolfo, and Kui W Mok. 1999. A data mining framework for building intrusion detection models. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*. IEEE, 120–132.

[31] Wei Liu and Sanjay Chawla. 2009. A game theoretical model for adversarial learning. In *Data Mining Workshops, 2009. ICDMW'09. IEEE International Conference on*. IEEE, 25–30.

[32] Wei Liu and Sanjay Chawla. 2010. Mining adversarial patterns via regularized loss minimization. *Machine learning* 81, 1 (2010), 69–83.

[33] Daniel Lowd and Christopher Meek. 2005. Adversarial learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 641–647.

[34] Blaine Nelson, Benjamin IP Rubinstein, Ling Huang, Anthony D Joseph, Steven J Lee, Satish Rao, and JD Tygar. 2012. Query strategies for evading convex-inducing classifiers. *Journal of Machine Learning Research* 13, May (2012), 1293–1332.

[35] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 372–387.

[36] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a defense to adversarial perturbations against deep neural networks. In *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 582–597.

[37] Roberto Perdisci, Guofei Gu, and Wenke Lee. 2006. Using an ensemble of one-class svm classifiers to harden payload-based anomaly detection systems. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*. IEEE, 488–498.

[38] Leonid Portnoy, Eleazar Eskin, and Sal Stolfo. 2001. Intrusion detection with unlabeled data using clustering. In *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001*. Citeseer.

[39] Marco Ramoni and Paola Sebastiani. 2001. Robust learning with missing data. *Machine Learning* 45, 2 (2001), 147–170.

[40] Arun A Ross, Karthik Nandakumar, and Anil K Jain. 2006. *Handbook of multibiometrics*. Vol. 6. Springer.

[41] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. 2016. Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. ACM, New York, NY, USA, 1528–1540. https://doi.org/10.1145/2976749.2978392

[42] Muazzam Siddiqui, Morgan C Wang, and Joohan Lee. 2008. A survey of data mining techniques for malware detection using file features. In *Proceedings of the 46th Annual Southeast Regional Conference on XX*. ACM, 509–510.

[43] David B Skillicorn. 2009. Adversarial knowledge discovery. *IEEE Intelligent Systems* 24, 6 (2009), 0054–61.

[44] Jungsuk Song, Hiroki Takakura, Yasuo Okabe, Masashi Eto, Daisuke Inoue, and Koji Nakao. 2011. Statistical analysis of honeypot data and building of Kyoto 2006+ dataset for NIDS evaluation. In *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*. ACM, 29–36.

[45] Choon H Teo, Amir Globerson, Sam T Roweis, and Alex J Smola. 2007. Convex learning with invariances. In *Advances in neural information processing systems*. 1489–1496.

[46] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2016. Stealing Machine Learning Models via Prediction APIs.. In *USENIX Security Symposium*. 601–618.

[47] Tich Phuoc Tran, Pohsiang Tsai, and Tony Jan. 2008. An adjustable combination of linear regression and modified probabilistic neural network for anti-spam filtering. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*. IEEE, 1–4.

[48] Yevgeniy Vorobeychik and Bo Li. 2014. Optimal randomized classification in adversarial settings. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 485–492.

[49] Xiao-Tong Yuan and Bao-Gang Hu. 2009. Robust feature extraction via information theoretic learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 1193–1200.

[50] Yan Zhou, Murat Kantarcioglu, Bhavani Thuraisingham, and Bowei Xi. 2012. Adversarial support vector machine learning. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1059–1067.