Università Ca' Foscari Venezia

Department of Environmental Sciences, Informatics and Statistics

Master's Degree
in *Computer science*

Robust Tree Ensemble against Adversarial Examples

Supervisor
Ch. Prof. Claudio Lucchese

Graduand
Federico Marcuzzi
Matricolation number
853770

Academic Year
2018/2019

Contents

Notations

x	variable
$\boldsymbol{x}$	vector
$\mathcal{D}$	input dataset
n, d	dataset dimension
$\mathcal{X}$	input spaces
$\mathcal{Y}$	output spaces
$\mathcal{H}$	hypothesis space
h	hypothesis (learner)
g	unknown target function
t	decision tree
$\mathcal{T}$	ensemble
f	feature
v	threshold
A	attacker
$\mathfrak{L}$	learning algorithm
$p(\cdot)$	probability density function
$p(\cdot\|\cdot)$	contitional probability density function
$\|\cdot\|$	size of the set

Abstract

Machine learning models are subject to attacks that undermine security and can generate unexpected behaviors. To the detriment of other machine learning algorithms, little has been done so far to create models based on decision trees that are robust to these attacks. The present thesis proposes a new method to train a model based on an ensemble of decision trees, robust to specific attacks. In particular, we developed a precise way of splitting the features of the dataset between the base-learners within the ensemble. This split guarantees a robustness by construction against evasion attack. The threat model we defined limits the attacker's strength to a maximum number of modifiable features. The constraint forces the attacker to generate adversarial example that differ from the original instance at most b features. Also together with the model we proposed two approximation algorithms to certify a lower bound robustness of the model. These approximation algorithms have significantly less computational complexity than testing the robustness of the model with brute-force attacks. We have compared the robustness of our model with *Random Forest* ensemble method and with some robust tree ensemble models taken from the state of the art of adversarial example. The experiments led to two important results. First, the comparison showed that the robustness of our model is higher than that of the other models tested. Secondly, the two robustness certification algorithms do not differ much from the real robustness of the model.

Introduction

In recent years machine learning (ML) has had an increase of its employment in many scenarios. Among these there are scenarios concerning the security of systems such as: intrusion detection, spam detection, fraud detection, self-driving cars, etc. Machine learning models applied in these scenarios must guarantee greater reliability than models applied in other contexts, sice they have to deal with critical situations. Unfortunately it has been realized that traditional ML models can be fooled by a malicious user (called attacker or adversary) who is able to subjugate the model for his own personal interests. This type of phenomenon is called adversarial machine learning (or adversarial learning).

The appearance of adversarial learning has brought to light vulnerabilities in existing machine learning models. These vulnerabilities can be more or less serious and can compromise the security, privacy and integrity of the system [1]. Normal ML models are subject to this type of attack because they are not designed to be robust to manipulations perpetrated by an intelligent entity. The perturbation generated by the attacker is different from a normal noise, so even models trained to manage noise may not guarantee safety against these attacks. Standard classification algorithms assume that the data generating process is independent from the classifier, but this is not true in the context of adversarial attack. The behavior of classical ML models under attack suggests that in the learning phase, they are not learning the semantics of data. As reported by Ian J. Goodfellow et al. in [25], the learning algorithm creates its own *"Potëmkin village"*, a model that works well on instances that occur naturally, but very badly when it encounters instances that have a low probability in the distribution.

In general, the most common attacks on ML models are evasion attacks and poisoning attacks. With an evasion attack (carefully-perturbed input samples), the adversary modifies instances to force the model to commit misclassification at test time [3, 38]. While with a poisoning attack, he contaminates the training set to create model malfunctions when it is used [8, 39]. The examples of system security mentioned above can be fooled by possible evasion, poisoning attacks in the following way: As reported in [18], a very simple example of adversarial learning is that applied to spam detection. Naïve Bayes has always been one of the best models to classify spam/ham emails. However, it has been shown that it's very easy to fool the model. If the attacker inserts so called good words (ham) in a spam email, or if he inserts typo or interruptions in spam words in order to camouflage them, the model is deceived and misclassifies the spam email as ham. In [41] it's shown how is possible to create glasses that allow the attacker to be misclassified as a different user in the system. For example, this type of attack allows a malicious user to circumvent a video surveillance system by impersonating a legitimate person. Furthermore, in [21, 8] it is shown how it is possible to edit a road signs (without precluding human recognition) in order to create a malfunction in a self-driving system. Finally, a customer of a bank can alter some information of his own state to get a loan even if he does not have the necessary conditions to obtain it.

Each of these attacks can have more or less serious consequences. It is therefore necessary to design models that are robust to these types of attacks. To counter this problem, some learning strategies have been developed which take into account the existence of an attacker during the training phase, in order to generate robust models. Several works have been proposed in literature to train robust linear classifiers [36], or deep neural networks [25, 45]. The latter are nowadays widely used in pattern recognition, computer vision, classification of images, etc. On the contrary, decision trees and tree ensemble have received very little attention so far. This is a serious lack sice decision trees are among the best methods for dealing with non-perceptual problems and their predictions and structure are human-understandable in terms of syntactic checks over domain features, which is particularly appealing in the security setting. Furthermore, the use of weak-learners like DT together with ensemble method techniques does not necessarily increase their robustness [27].

Contributions of this work

The main contribution of this thesis is the development of an ensemble learning algorithm, based on decision trees, robust to evasion attack. The algorithm we proposed is called Feature Partitioned Forest and the resulting ensemble is a binary classifier, robust by construction to attacks up to a certain strength. Given an opponent who transforms a sample $\boldsymbol{x}$ in an evading instance $\boldsymbol{x}'$ such that $\| \boldsymbol{x} - \boldsymbol{x}' \|_0 \leq k$, our algorithm guarantees the majority of the forest is not affected by the attack. In an adversarial learning scenario it is fundamental to define the attacker in a threat model that limits his strength. In L_0-norm, k represents the maximum number of features the attacker can modify. Through this value we have defined and limited the strength of the attacker. The robustness by construction of FPF is based on a particular sampling of the features, where it randomly equi-partition the set of features, and train each tree on a distinct feature partition. The number of partitions is closely related to the strength of the attacker and ensure that the attacked features appear in less than half of the forest. This way the attacker cannot attack the majority of the trees. To show the advantage obtained with our algorithm,we have compared FPF with Random Forest [10] and Random Subspace Method [29, 5] and we showed empirically that our strategy is more robust than its competitors. Specifically, we showed that on Breast_Cancer dataset, the difference between the accuracy under attack of FPF with respect to RF and RSM was 0.072 and 0.021 respectively, with attacks on two features. For the Spam_Base dataset, the difference between the FPF's accuracy under attack and the one obtained by RF and RSM was 0.254 and 0.004 respectively, attacking one features. Instead with attacks on two features, the difference was 0.591 and 0.081 respectively. Finally in the Binary_Wine dataset the difference between FPF's accuracy under attack and the one of RF and RSM was 0.607 and 0.099 respectively, attacking two features. In all datasets, our model has achieved greater robustness than its competitors. In addition, we have provided two certificates called accurate lower-bound and fast lower-bound that efficiently assesses the minimal accuracy of a forest on a given dataset, avoiding the costly computation of evasion attacks. For both lower bounds we have shown empirically how the estimated accuracy is very close to the real one. For example, we showed empirically that for an attacker with budget $k = 1$, both ALB and FLB lower bounds made an relative error compared to the real accuracy under attack of 1.6% on Breast_Cancer, 1.7% on Binary_Wine and 8% on Spam_Base. All the code developed to conduct the experiments of this thesis can be found at the following link: github.com/FedericoMarcuzzi/Feature-Partitioned-Forest.

Thesis structure

- **Chapter 1: Machine Learning.** This chapter deals with the basic elements of machine learning. The definitions of *supervided learning*, *Decision Tree* and *ensemble methods* are given. Among the latter, particular importance is given to *Random Forest*, *Bagging*, and *Gradient Boosted Decision Tree.*

- **Chapter 2: Adversarial Machine Learning.** In this chapter we defined the taxonomy of possible adversarial machine learning attacks. It is defined what an attacker is and his properties (goal, knowledge, capability and strategy). Finally, the most common attacks and some of their countermeasures are given.

- **Chapter 3: State of the art.** In this chapter we reported the state of the art of robust training algorithms. In particular, we focused on the algorithms related to decision trees.

- **Chapter 4: Feature Partitioned Forest.** In the fourth chapter of the thesis we illustrated our ensemble learning algorithm FPF. We also defined the attacker model, the FPF properties and the two lower bounds to certify the minimum performance of the model.

- **Chapter 5: Evaluation methodology.** The fifth chapter collects the experimental setting used to compare the robustness of FPF with other algorithms. In the experiments we also took into account the training parameters of each algorithm, such as: number of trees, maximum number of leaves, number of features to be protected (for FPF) and other specific parameters. Experiments are also conducted to empirically demonstrate that the proposed certificates are indeed very accurate.

- **Chapter 6: Experimental results.** In this chapter we summarized the results obtained from the experiments conducted on the models and certificates. For each model and certificate, we discussed its strengths and weaknesses.
- **Chapter 7: Conclusions.** We briefly summarized the work done in this thesis and the contribution we made with the design of the FPF algorithm and its certificates of robustness.
- **Chapter 8: Future Works.** In the last chapter we reported some possible ideas to continue the development of the FPF algorithm and make it even more effective.

Chapter 1

Machine learning

This chapter introduces the basic concept of machine learning, the notation used in this thesis, and some fundamental algorithm for this work such as *Decision Tree*, *Random Forest*, *Bagging* and *Gradient Boosted Decision Tree.*

1.1 Supervised Learning

Supervised learning is a machine learning technique that maps an instance to a label. Informally, we can say that supervised learning is a technique that through some correctly labeled objects, finds a function (also known as *learner*) that best discriminates them. Later through this function it is possible to assign the correct labels to new objects never seen before. Let $\mathcal{X} \subseteq \mathbb{R}^d$ be a d-dimensional vector space of real-valued features and the elements $\boldsymbol{x} \in \mathcal{X}$ the so called *instances*, with $\boldsymbol{x} = (x^{(1)}, \ldots, x^{(d)})$. Each instance $\boldsymbol{x}$ is associated with a label $y \in \mathcal{Y}$, with $\mathcal{Y}$ the output space. The mapping between the instances in $\mathcal{X}$ and the labels in $\mathcal{Y}$ is done by an unkown function $g : \mathcal{X} \to \mathcal{Y}$, called the *target* function. The function g is called target because is the function that the learning algorithm tries to approximate. Given the hypothesis set $\mathcal{H}$, the supervised learning algorithm finds the best function $h \in \mathcal{H}$ that best approximates the function g. The learning algorithm exploits a *training* set, $\mathcal{D} = \{(\boldsymbol{x}_1, g(\boldsymbol{x}_1)), \ldots, (\boldsymbol{x}_n, g(\boldsymbol{x}_n)\}$, with $n = |\mathcal{D}|$, which is a set of (*instance,correct label*) pairs. In accordance with statistical learning theory, the function $\hat{h}$ that best approximates g, can be found by means of *empirical risk minimization* [47]. Given $\mathcal{D}$ and $\mathcal{H}$, the empirical risk is defined as a loss function $\mathcal{L} : \mathcal{H} \times (\mathcal{Y} \times \mathcal{X})^n \to \mathbb{R}^+$. $\mathcal{L}$ is a loss function that measures the cost of an erroneous prediction performed by $\hat{h}(\boldsymbol{x})$ with respect to g. To find the hypothesis that minimizes the empirical risk the following optimization problem must be solved:

$$\hat{h} = \arg\min_{h \in \mathcal{H}} \mathcal{L}(\mathcal{Y}, h(\mathcal{X})) \tag{1.1}$$

with $\hat{h}$ the hypothesis with the lowest erroreous prediction cost. Finally, the $\mathcal{L}$ function can also be obtained by aggregating the *instance-level loss*, $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$, then $\mathcal{L}$ can be defined as $\sum_{(\boldsymbol{x},y)\in\mathcal{D}} \ell(y, h(\boldsymbol{x}))$.

1.2 Decision Trees

A Decision Tree (DT) is one of the most classic machine learning models, which takes its name from its tree structure. A DT has two main components: the internal nodes and leaves. Each node in the tree performs a features test (or split) on the input data. In particular, each node is associated with a certain features f on which to perform the test and a v value that is the condition. The data that crosses a node will be subdivided into subsets according to the response of the test. Each leaf of the tree is associated with a label (in case of classification) or a numerical value (in case of regression). During the prediction phase, the instances will be assigned to the label/value of the leaf they reach. The split can be performed on both categorical and numerical features. In the case of a categorical feature, chosen the category with which to discriminate the elements (test category), the set is divided according to whether or not an element belong to it. In the case of numeric features, the set will be divided into elements whose value of the tested feature is lower than the threshold and those in which it is not. In Figure 1.1 there is an example of a tree with both categorial split (n_1) and numeric split (n_2). The above

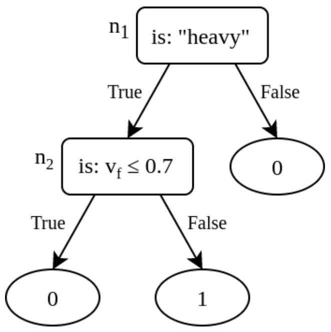


Figure 1.1: Decision tree example, with both numeric and categorical features.

is more about the prediction phase of the decision tree, but another key aspect of the model is how it is created. The creation of a DT is usually done recursively, and at each step of the recursion, the best split point is chosen. The split divides the input set into sub-sets, which will be the input for subsequent recursions. With each recursion the size of the input set decreases. The recursion continues until the base case is reached (stop criteria). To stop the growth of the tree there are several stop criteria such as: **max depth** stops the growth when the tree reaches a certain height, **max leaves** stops the growth when the maximum number of leaves is reached or **purity** which ends the growth when the majority class in one leaf is greater than a certain threshold compared to the others. The core of the learning phase is how to choose the best split. The best split can be defined as a pair (f^*, v^*), that represent the feature and the value that best divide the set $\mathcal{D}$. As reported in [48], different criteria have been used to find the best split. Below are the definitions of the most common criteria:

Information Gain The information gain criterion can be used to select a split using the concept of entropy. Let $\mathcal{D}$ be the input to a node, its entropy is defined as:

$$H(\mathcal{D}) = -\sum_{y \in \mathcal{Y}} P(y|\mathcal{D}) \log P(y|\mathcal{D}). \tag{1.2}$$

If $\mathcal{D}$ is divided into subsets $\mathcal{D}_1, \dots, \mathcal{D}_k$, its entropy could be reduced. The amount of reduced entropy is called information gain and is defined as follows:

$$IG(\mathcal{D}; \mathcal{D}_1, \dots, \mathcal{D}_k) = H(\mathcal{D}) - \sum_{i=1}^{k} \frac{|\mathcal{D}_i|}{|\mathcal{D}|} H(\mathcal{D}_i). \tag{1.3}$$

At this point, to find the best split (f^*, v^*), which generates the highest entropy, it is sufficient to solve the following maximization problem:

$$(f^*, v^*) = \operatorname*{arg\,max}_{f \in \mathcal{F}, v \in \mathcal{V}_f} IG(\mathcal{D}; \text{split}(\mathcal{D}, f, v)), \tag{1.4}$$

where the split function divides $\mathcal{D}$ in k different subsets $\mathcal{D}_1, \dots, \mathcal{D}_k = \text{split}(\mathcal{D}, f, v)$, according to the feature $f \in \mathcal{F}$, with $\mathcal{F}$ the set of possible features in $\mathcal{D}$ and the value $v \in \mathcal{V}_f$, with $\mathcal{V}_f$ the set of possible value for v with respect to the features f. However the information gain has the problem of favouring features that have many values, to the detriment of the final generalization of the model. Considering to dealing with categorical variables, if a feature has a different value for each instance of the training set, taking it as split leads to a high information gain, since it is possible to perfectly divide all the training instances. But the choice of this feature causes a bad generalization of the model and would be useless in predicting unseen instances. By choosing this feature, each test node divides the training set into smaller and more pure subsets (belonging to the same class). As a consequence there will be a large number of leaves, each with one instance. In this case the fact that the leaves have a high purity is not relevant, the model overfitted on training instances.

Gain Ratio To prevent features with many values from being privileged, another selection criterion called gain ration can be used:

$$P(\mathcal{D};\mathcal{D}_1,\ldots,\mathcal{D}_k) = IG(\mathcal{D};\mathcal{D}_1,\ldots,\mathcal{D}_k)\cdot\left(-\sum_{i=1}^{k}\frac{|\mathcal{D}_i|}{|\mathcal{D}|}\log\frac{|\mathcal{D}_i|}{|\mathcal{D}|}\right)^{-1}, \tag{1.5}$$

The gain ratio is a variant of the information gain, which taking normalization on the number of feature values and childern/partitions of the dataset. The feature that has the highest gain ratio, among features with better-than-average information gains, is chosen for the split. In this way features with many values have the same importance as features with few values. So to find the pair (f^*, v^*), just solve the maximization 1.4 but on gain ratio P instead of information gain IG.

Gini Index Finally, the definition of Gini index is given, another criterion that can be used to find the best split:

$$IG_{gini}(\mathcal{D};\mathcal{D}_1,\ldots,\mathcal{D}_k) = I(\mathcal{D}) - \sum_{i=1}^{k}\frac{|\mathcal{D}_i|}{|\mathcal{D}|}I(\mathcal{D}_i), \tag{1.6}$$

where

$$I(\mathcal{D}) = 1 - \sum_{y\in\mathcal{Y}}P(y|\mathcal{D})^2. \tag{1.7}$$

Again, to find the best split (f^*, v^*), just solve the maximization problem 1.4 on G_{gini} instead of IG.

Using only the search for the best split does not guarantee to create a decision tree that generalizes well. In particular, if two decision trees are compared, one with perfect performance on the training set and another with lower performance, it can often be observed that the first has lower performance on data never seen before. This phenomenon is called overfitting, that is when the model adapts too much to the training set, also learning from the noise that are exchanged as underlying truth. As a consequence it produces a model that generalizes badly. To reduce this problem, a technique called **pruning** can be applied. Pruning consists in removing some branches that are not necessary and is divided into pre-pruning and post-pruning. **Pre-pruning** tries to remove the branches while the tree is growing. This can be done by using a validation set. The tree branch is not expanded if its growth would result in an increase in validation-error. **Post-pruning** is applied after the whole tree is created. The entire tree is inspected to see which branches can be removed. Also in this case with the use of a validation set one branch can be cut if the validation-error decreases after its removal.
This thesis is focused on binary decision trees, that handles only numeric features

and performs binary classification tasks. As defined in [12], a inductively definition of tree is given. A DT t is either a leaf or a non-leaf (internal node). A leaf is defined as $\lambda(\hat{y})$ for some labels $\hat{y} \in \{0, 1\}$. Instead a non-leaf is defined as $\sigma(f, v, t_l, t_r)$, where $f \in [1, d]$ is a feature of the dataset, $v \in \mathbb{R}$ is the threshold for the features f and t_l,t_r are rispectively the decision trees to the left and right branch of t. During the classification phase, an instance $\boldsymbol{x}$ traverses the tree t from the root to a leaf $\lambda(\hat{y})$, which returns the predicion $\hat{y} = t(\boldsymbol{x})$. The traversal from the root to a leaf is made trought the internal node. Each instance $\boldsymbol{x}$ at the node $\sigma(f, v, t_l, t_r)$ performs the feature test $x^{(f)} \leq v$. If the answer is positive the instance will go into the left branch, otherwise in the right branch. So at each feature test, the instance goes through a single branch at a time and ends in a single leaf.

1.3 Ensemble Methods

This section gives an overview of what an ensemble method is, its potential and it is focused on Random Forest and Gradient Boosting Decision Tree. Usually the learning approach involves only a single learner to solve a given problem. Ensemble methods, on the other hand, combine several learners to carry out the same task. This type of learing is called ensemble learning. The learners inside the enemble are called base-learners (or weak-learners) and are trained through a base-learning algorithm. In general, the creation of an ensemble takes place in two steps. In the first step the base-learners are created, while in the second they are combined through a combination strategy. It is possible to use the same base-learning algorithm to create base-learners of the same type and obtain a homogeneous ensemble. Or is possible to use a multiple base-learning algorithms which generate heterogeneous ensembles. To get a good ensemble, learners must be as accurate as possible and as different as possible from each other. Once the learning algorithm is defined, there are two paradigms on how to create the ensemble, which are sequential and parallel ensemble methods. With the sequential ensemble methods the base-learners are created sequentially. In this way it is possible to exploit the dependence between the learners and try to increase the performance of the model by adding a new learner (Gradient Boosting). The parallel ensemble methods, on the other hand, imply that the base-learners are created in parallel and their independence can be exploited, since the ensemble's error can be reduced by combining independent learners (Bagging). Unfortunately, in practice it is not possible to create good independent learners, due to the finite size of the training dataset. However it is possible to create less dependent base-learners by introducing randomness in the learning algorithms. The computational complexity of generating an ensemble is not greater than generating a single base-learner, as it is sufficient to train more base-learners and usually the cost of combining them

toghether is very low.
The power of the ensembles lies in two main facts. First, combining weak-learners usually leads to a better generalization than an indivudual learner. Second, creating many weak-learners and combining them together is much simpler than creating a very strong one.

1.3.1 Random Forest

Random Forest [10] is one of the most famous ensemble method, and is based on decision trees. The training algorithm does not differ much from Bagging 1.3.1, except for a random component in the trees construction. In particular, in the DT construction phase, the choice of the best split is not made on all the features of the dataset $\mathcal{D}$, but on a randomly chosen subset of features, of cardinality k. This random part helps generalization because each node of each tree has a different subset of features on which to split the data. It is possible that each base-learner will experience a decrease in performance since it has a smaller search space, but it is mitigated when inserted into the ensemble. The complete Random Forest algorithm is given at 1.

Bagging

To better understand the functioning of the random forest it is also necessary to explain what Bagging is and how it works. The full name *Bootstrap AGGregatING* [9], suggests that the main components of the algorithm are boostrap and aggregation. **Boostrap**: From the above, the basic combination of independent learners generates a more efficient model. Model independence can be achieved by simply training each base-learner on a disjointed subset of the trainig set. But since the size of the training set is not infinite, each base-learner will only have a small portion of datasets to learn from and this will lead to bad performance. Boostrap sampling [20], allows to have a trade-off between different instances for learner and quantity. Specifically with n instances in the dataset, boostrap sampling can generate samples of size n by performing a sampling with replacement. In this way the original instances will never be all included in the same sample, but there will be some repetitions and some exclusions of instances. Boostrap samplig can be repeated r times and then get r sample of size n. For each sample a base-learner can be trained.
More in detail, as reported in [48], the boostrap sampling gives Bagging an additional advantage. With n training instances the probability that the i-th instance will be extracted $0, 1, 2 \ldots$ is approximately a distribution of Poisson with $\lambda = 1$. Thus the probability i-th example will occur at least once is $1 - (1/e) \approx 0.632$. Therefore each sample will not have about 36.8% of instances of the orignal dataset.

Algorithm 1 RandomForest

```
Require: A training set $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, features set $\mathcal{F}$ and base-learning algorithm $\mathfrak{L}$
 1: function RandomForest($\mathcal{D}, \mathcal{F}$)
 2:     return Bagging($\mathcal{D}, \mathcal{F}$, RandomForestTree)
 3: end function

 4: function Bagging($\mathcal{D}, \mathcal{F}, \mathfrak{L}$)
 5:     $\mathcal{T} \leftarrow \emptyset$
 6:     for $i = 1$ to $r$ do
 7:         $S^{(i)} \leftarrow$ a boostrap sample from $D$
 8:         $t_i \leftarrow \mathfrak{L}(S^{(i)}, \mathcal{F})$
 9:         $\mathcal{T} \leftarrow \mathcal{T} \cup t_i$
10:     end for
11:     return $\mathcal{T}$
12: end function

13: function RandomForestTree($\mathcal{D}, \mathcal{F}$)
14:     if all instances in the same class $\vee\ \mathcal{F} = \emptyset$ then
15:         return $\lambda(\hat{y}(\mathcal{D}))$
16:     end if
17:     $t \leftarrow \sigma(f, v, t_l, t_r)$
18:     $\hat{\mathcal{F}} \subseteq \mathcal{F} : |\hat{\mathcal{F}}| = k \wedge k \ll |\mathcal{F}|$
19:     $(t.f, t.v) \leftarrow$ split($\mathcal{D}, \hat{\mathcal{F}}$)
20:     $t.t_l \leftarrow$ RandomForestTree($\mathcal{D}_{t.f} \leq t.v, \mathcal{F}$)
21:     $t.t_r \leftarrow$ RandomForestTree($\mathcal{D}_{t.f} > t.v, \mathcal{F}$)
22: end function
```

This 1/3 of unseen instances for each base-learner allows the creation of a surrogate model independence by training each base-learner on different subset of the trainig instances. **Aggregation:** Very simply Bagging uses voting for classification and averaging for regression. Voting means taking the most frequent label returned by learners. Instead with averaging the result is calculated by performing the average of the predictions of all learners.

1.3.2 Gradient Boosted Decision Tree

The Gradient Boosting Decision Tree (GBDT) is an ensemble methods based on the Gradient Boosting construction algorithm (GB) where the decison tree learning algorithm is used as a base-learner. Jerome H. Friedman in [23, 24] has formalized the GB and GBDT algorithms in a generalized way, without binding too much to the loss function used.

Gradient Boosting The GB is an iterative meta-algorithm that indicates how to combine the weak-learners to create a strong learner. In particular, each iteration, it creates a new base-learner on the errors generated by the strong learner (the esemble). The base-learner is added to the ensemble to improve its performance. Learners are functions and by adding them together it is possible to create a stronger function that better fit the data. GB is a supervised learning algorithm and as explained in 1.1, given a training set $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_n, y_n)\}$, its goal is to find the function $\hat{h}(\boldsymbol{x})$, that best approximates the target function g. This means searching for the $\hat{h}(\boldsymbol{x})$ function that commits the lowest number of errors. This can be expressed as a minimization of the expected value $E_{(\boldsymbol{x},y)\in\mathcal{D}}\mathcal{L}(y, h(\boldsymbol{x}))$ as follows:

$$\hat{h}(\boldsymbol{x}) = \arg\min_{h(\boldsymbol{x})} E_{(\boldsymbol{x},y)\in\mathcal{D}}\mathcal{L}(y, h(\boldsymbol{x})). \tag{1.8}$$

Solving the optimization problem 1.8, can be proebitive, so to overcome this problem, boosting approximates the $\hat{h}(\boldsymbol{x})$ function through additive expansion defined as follows:

$$h(\boldsymbol{x}) = \sum_{m=0}^{M} \beta_m \mathfrak{L}(\boldsymbol{x}; \boldsymbol{a}_m), \tag{1.9}$$

where $\mathfrak{L}(\boldsymbol{x}; \boldsymbol{a}_m)$ is the weak-learner, $\boldsymbol{a}_m = \{a_m^{(1)}, a_m^{(2)}, \dots\}$ is the vector of the model parameters and b_m is the expansion coefficient. At each iteration $m = 0, \dots, M$ of the algorithm, $\{b_m\}_0^M$ and $\{\boldsymbol{a}_m\}_0^M$ are calculated and used in the m-th weak-learner training. Boosting algorithm uses a strong learner initial guess $h_0(\boldsymbol{x})$ from which to start the expansion of the ensemble:

$$(\beta_m, \boldsymbol{a}_m) = \arg\min_{\beta,\boldsymbol{a}} \sum_{i=1}^{N} \mathcal{L}(y_i, h_{m-1}(\boldsymbol{x}_i) + \beta\mathfrak{L}(\boldsymbol{x}_i; \boldsymbol{a})). \tag{1.10}$$

The evolution of the strong learner occurs by adding to $h_{m-1}(\boldsymbol{x})$ the weak-learner $\beta_m\mathfrak{L}(\boldsymbol{x}; \boldsymbol{a}_m)$, created at the m-th iteration, generating the new strong learner $h_m(\boldsymbol{x})$:

$$h_m(\boldsymbol{x}) = h_{m-1}(\boldsymbol{x}) + \beta_m\mathfrak{L}(\boldsymbol{x}; \boldsymbol{a}_m). \tag{1.11}$$

To solve the optimization problem 1.10, for a differentiable albitrary loss function $\mathcal{L}(y, h(\boldsymbol{x}))$, a two-step procedure is proposed in [23]. The first step involves fitting $\mathfrak{L}(\boldsymbol{x}; \boldsymbol{a})$ with the least-squares $\boldsymbol{a}_m$ of the current preudo-residual $\tilde{y}_{im}$.

$$\boldsymbol{a}_m = \arg\min_{\boldsymbol{a},\rho} \sum_{i=1}^{N} [\tilde{y}_{im} - \rho\mathfrak{L}(\boldsymbol{x}_i; \boldsymbol{a})]^2, \tag{1.12}$$

where $\tilde{y}_{im}$ are the current pseudo residual defined as the derivative of the loss funtion:

$$\tilde{y}_{im} = -\left[\frac{\partial \mathcal{L}(y_i, h(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)}\right]_{h(\boldsymbol{x})=h_{m-1}(\boldsymbol{x})} \tag{1.13}$$

In 1.13 gradient boosting minimizes pseudo-residuals, which are the gradient of the loss function to be minimized. This means that the algorithm applies the gradient descent to solve the minimization problem. The second step is to calculate the optimal coefficient β_m for the model $\mathfrak{L}(\boldsymbol{x}; \boldsymbol{a}_m)$ as follows:

$$\beta_m = \arg\min_{\beta} \sum_{i=1}^{N} \mathcal{L}(y_i, h_{m-1}(\boldsymbol{x}_i) + \beta \mathfrak{L}(\boldsymbol{x}_i; \boldsymbol{a}_m)). \tag{1.14}$$

This approximation based on the least-squares 1.12 trains a strong learner through the GB, avoiding an onerous problem of function optimization 1.10, performing a single parameter optimization (β_m) 1.14 with respect to the loss function $\mathcal{L}$.

Gradient Boosting Decision Tree What has been shown above is a general approach for creating an ensemble based on gradient boosting. If $\mathfrak{L}$ is a decision tree, then the algorithm is called Gradient Boosting Decision Tree. The GBDT is competitive, interpretable, both for classification and regression, especially for mining less than clean data. Friedman [24] has also proposed a generalized version of GBDT on a generic loss function $\mathcal{L}$. Let $\mathfrak{L}(\boldsymbol{x}; \boldsymbol{a})$ be a regression tree with L leaves, called L-leaves_learner. At each iteration of gradient boosting, L-leaves_learner divides the instance space $\boldsymbol{x}$ into L disjoint regions denoted as $\{R_{lm}\}_{l=1}^{L}$ and predicts a constant for each region.

$$\mathfrak{L}(\boldsymbol{x}; \{R_{lm}\}_1^L) = \sum_{l=1}^{L} \bar{y}_{lm} 1(\boldsymbol{x} \in R_{lm}), \tag{1.15}$$

where

$$\bar{y}_{lm} = \frac{\sum_{\boldsymbol{x}_i \in R_{lm}} \tilde{y}_{im}}{|R_{lm}|}. \tag{1.16}$$

The values $\bar{y}_{lm}$ is the mean of the pseudo-residual $\tilde{y}_{im}$ computed with 1.13, of the instances fallen in the region R_{lm}. Since $\mathfrak{L}$ is a tree, the parameters of L-leave_learner are the splitting variables and the splitting points for each variable. For each iteration m, these parameters define how the instances will be divided into the regions $\{R_{lm}\}_{i=1}^{L}$. Furthermore the use of the decision tree allows to reduce the minimization problem 1.14 to a simple locate estimate based on the loss function $\mathcal{L}$, through the predictions $\bar{y}_{lm}$ of the model produced by 1.15.

$$\gamma_{lm} = \arg\min_{\gamma} \sum_{\boldsymbol{x}_i \in R_{lm}} \mathcal{L}(y_i, h_{m-1}(\boldsymbol{x}_i) + \gamma). \tag{1.17}$$

Thanks to this, to create the next strong learner h_m, it is enough to update the result of each leaf (region) of the learner h_{m-1} with the new values estimated at the iteration m.

$$h_m(\boldsymbol{x}) = h_{m-1}(\boldsymbol{x}) + \nu \cdot \gamma_{lm} 1(\boldsymbol{x} \in R_{lm}), \tag{1.18}$$

where $0 < \nu \leq 1$ is the *shrinkage* parameter. GB is subject to overfitting, so to avoid this problem the *incremental shrinkage* is used. Incremental shrinkage consists of adding a multiplicative constant ν at each adjustment of the regions of $h_{m-1}(\boldsymbol{x})$. In [23] it is empirically shown that a value of $\nu \leq 0.1$ leads to a better generalization. In fact, it is a known best practice to try to start with a value of $\nu = 0.1$ and to gradually lower it during the learning phase. Finally in [23] it is possible to find specific algorithms based on this GBDT template for different loss criteria, including, least-squares: $\mathcal{L}(y, F) = (y - F)^2$ and least-absolute-deviation: $\mathcal{L}(y, F) = |y - F|$ for $y \in R^1$ (regression) and negative binomial log-likelihood: $\log(1 + e^{-2yF)}$, with $y \in \{-1, 1\}$ (classification). The following is the pseudo-code 2 of the Generalized GBDT reported in [24].

Algorithm 2 GeneralizedGBDT

```
Require: A set of the training instances $\boldsymbol{x} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, the corresponding
    outputs $y = \{y_1, \ldots, y_n\}$ and loss function $\mathcal{L}$
 1: function GeneralizedGBDT($\boldsymbol{x}, \mathcal{L}$)
 2:     $h_0(\boldsymbol{x}) \leftarrow \arg\min_\gamma \sum_{i=1}^{N} \mathcal{L}(y_i, \gamma)$
 3:     for $m = 1$ to $M$ do
 4:         $\tilde{y}_{im} \leftarrow -\left[\frac{\partial \mathcal{L}(y_i, F(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)}\right]_{h(\boldsymbol{x}) = h_{m-1}(\boldsymbol{x})}, i = 1, \ldots, N$
 5:         $\{R_{lm}\}_{l=1}^{L} \leftarrow \mathsf{L} - \mathsf{leaves\_learner}(\{\tilde{y}_{im}, \boldsymbol{x}_i\}_{i=1}^{N})$
 6:         $\gamma_{lm} \leftarrow \arg\min_\gamma \sum_{\boldsymbol{x}_i \in R_{lm}} \mathcal{L}(y_i, h_{m-1}(\boldsymbol{x}_i) + \gamma)$
 7:         $h_m(\boldsymbol{x}) = h_{m-1}(\boldsymbol{x}) + \nu \cdot \gamma_{lm} 1(\boldsymbol{x} \in R_{lm})$
 8:     end for
 9:     return $h_m(\boldsymbol{x})$
10: end function
```

Summary

In this first chapter we have introduced some basic machine learning concepts, fundamental for understanding the work we have carried out in this thesis. Below we have summarized the main concepts covered in this chapter.

- **Supervived learning.** The first part of the chapter focused on supervised learning, the learning technique on which the models used in this thesis are based. As we have seen, given the training dataset, supervised learning uses empirical risk minimization to find the hypothesis $\hat{h}$ that best approximates the target function g.

- **Decision Trees.** Then we gave the definition of Decision Trees learning algorithm. We analyzed the various strategies for choosing the best spliting point: **Information Gain**, **Gain Ratio** and **Gini Index**. We have seen some cases of stop criteria such as: **max depth**, **max leaves** and **purity** and finally the **pruning** strategies and their usefulness.

- **Ensemble methods.** In this section we have given a general introduction to the ensemble methods and the **base-learners** that compose them. Each ensemble is built in two steps, the creation of the base-learners through a **base-learning algorithm** and the combination of their predictions through a **combination strategy**. We have explained how it is possible to exploit the dependence and independence between learners respectively through a sequential and parallel construction. Finally we gave the difference between homogeneous and heterogeneous ensembles.

- **Random Forest.** The section of the ensemble method Random Forest is the most important of the chapter because RF is one of the learning algorithm used to compare our algorithm. In this section we have explained **Bagging** and **boostrap samplig** algorithms, useful concepts also in the following chapters. Knowledge of the structure of RF has been fundamental to conduct the experiments and motivate the results.

- **Gradient Boosted Decision Tree.** Finally in the last section of this chapter we defined the **Gradient Boosting** ensemble method and its adaptation to decision trees, the Gradient Boosted Decision Tree (GBDT). We have given a generalized version of the GBDT applied to a generic loss function. GBDT has become the state-of-the-art of models based on decision trees thanks to the high performance it can achieve. Intuitively in a GBDT each tree is trained to correct the error committed by the previous one.

Chapter 2

Adversarial Machine Learning

In this section it exposes the main themes of adversarial machine learning, fundamental for understanding the work done in this thesis. Adversarial machine learning is a technique that is used in the context of machine learning to subjugate a model to behave incorrectly. Nowadays, machine learning is widely used in different contexts, ranging from intrusion detection, spam detection, fraud detection, etc.. The first reference to adversarial machine learning in [18] dates back to 2004 . The appearance of adversarial learning has brought to light vulnerabilities in existing machine learning models. These vulnerabilities can be more or less serious and can compromise the security, privacy and integrity of the system. Because of these vulnerabilities in machine learning models, an arms race has arisen for the development of robust models for different types of attacks and adversary's models. This section is organized as follows:

- Section 2.1 collects a taxonomy of attacks and their meaning.
- Section 2.2 defines the attacker's model (goals, knowledge of the model to attack, capability and strategy).
- Section 2.3 introduces the main types of attacks with particular attention to evasion attacks 2.3.1.
- Section 2.4 exposes some general countermeasures used nowadays against these attacks.

2.1 Attack taxonomy

In the literature [30, 4, 7] a taxonomy of potential attacks against pattern recognition systems has been provided. The attack consists of three fundamental features that are: the *influence* that the attack has on the classifier, the type of *security violation*

that the attack causes and finally the *specificity* of the attack.
The feature *influence* is divided into two types of attack, the **causative** and the **exploratory**.

- **causative:** In a causative attack the adversary interferes with the learning phase by altering the training set. For example, the attacker can inject malicious instances to reduce the performance of the final model.
- **exploratory:** In an exploratory attacks the adversary tries to extract information from an already trained model. Through this information it is possible to create instances that fools the model without modifying the learning algorithm.

The second features, *security violation*, is divided into three types of violation: **integrity**, **availability** and **privacy** violation.

- **integrity violation:** A violation of integrity means that an attacker tries to classify as good malicious instances.
- **availability violation:** An availability violation occurs when the attack prevents legitimate users from accessing the service. It is a denial-of-service (DoS) attack, the attacker forces the model to make a large number of errors, so that the normal service is compromised.
- **privacy violation:** With a privacy violation attacks the adversary can infer confidential user information (*e.g.,* example biometric information).

Finally the feature *specificity* can be **targeted** or **indiscriminate**.

- **targeted:** The effect of a targeted attack is aimed only at one instance (or a restricted set of instances).
- **indiscriminate:** In an indiscriminate attack, the attacker only wants to create a false negative, without focusing on a specific class of points.

2.2 Adversary's Model

In [8, 7] they proposed an attacker model divided into four basic points. First, the attacker goal's: what he wants to attack and what he wants to achieve. Second, the knowledge he has abount the target system. Third, his capability to generate the attack and fourth, his strategy. Having a definition of the attacker is fundamental for the development of a robust model. Often the robustness of a model is tied to the model of the attacker for which it was designed.

2.2.1 Adversary's Goal

The attacker's goal is defined on the two features security violation and specifity. Respectively, the first identifies the type of violation that the attacker wants to apply against the system (integrity, availability or privacy). The second instead identifies the portion of the dataset on which to perform the attack (a restricted subset or the whole dataset). The attacker's goal is to maximize the damage, and therefore it can be formulated as an optimization problem (optima attack strategy). Table 2.1 highlights the relationship between features specifity and security violation.

Table 2.1: Adversary's goals, relationship between features specifity and security violation. T and I are rispectively Targeted and Indiscriminate attack. Part of the information in the table was taken from [1]

		Integrity	Availability	Privacy
Causative	T	Permit a specific intrusion	Create sufficient errors to make system unusable for one person or service	Sign into the system as a specific person
	I	Permit at least one intrusion	Create sufficient errors to make learner unusable	Sign into the system as an arbitrary person
Exploratory	T	Find a permitted intrusion from a small set of possibilities	Find a set of points misclassified by the learner	Access information about a specific person
	I	Find a permitted intrusion		Access information of an arbitrary person

2.2.2 Adversary's Knowledge

The adversary's knowledge about the model is defined on the amount of information he has on how the model was trained. This information is: the dataset used to perform the training, how instances are represented in features, which learning algorithm is used, which decision function is used, the model parameters and finally the output (or feedback) returned by model in the test phase. Based on the amount of information the attacker has on the system, it is possible to define three main attack scenarios as reported by Biggio and Roli in [8]:

White Box Attack It is said that an attack is white-box or that the attacker has a perfect-kowledge, when he knows all the information used for training the model, including its parameters. This scenario allows the adversary to perform the strongest attack. In this context, creating attacks is easier.

Grey Box Attack In a gray box attack (also known as limited-knowledge) the information known to the attacker is only the learning algorithm used to build the model and the features representation. In this case the attacker does not know the training dataset or even the parameters of the model he wants to attack, but he can still use a surrogate dataset. In a surrogate dataset instances ideally come from the same underlying data distribution, and by querying the model with these instances, the attacker can extrapolate useful information. The model returns feedback that is used to label surrogate data. In this way the attacker can modify a surrogate instance in input to the model until the returned label is the one desired (or different form the original). Alternatively, the attacker can query the target model by sending surrogate data, and with the returned labels create a surrogate model on which to generate attacks. This attack is discussed in detail in the transferability section 2.3.4

Black Box Attack Finally, there is the zero-knowledge scenario or black box attack in which the attacker has no knowledge of the learning algorithm, the training dataset and even the features representation of the system. Despite this scenario it is still possible to perform an attack if the adversary can interrogate the model and get feedback. The attacker can also find further information by simply reasoning about the purpose of the model. If the task of the model is to classify road signs, he can assume that the training set concerns this type of object (road signs, traffic lights, pedestrian strips, etc.). If the attacker infers how the training set is made, he can perform an attack similar to the gray box attack (surrogate dataset, surrogate model, ect.). It is important to point out that it is possible to base security on secrets for the adversary, but it is never recommended to base the entire security of the model only on this. Indeed, security through obscurity is a false security, it is a risk to entrust the robustness of the model to secrets that are not reasonable or that can be inferred. Therefore the use of the adversary's ignorance must necessarily be accompanied by a solid design of the model and always assume that the attacker knows at least the features representation and learning algorithm.

2.2.3 Adversary's Capability

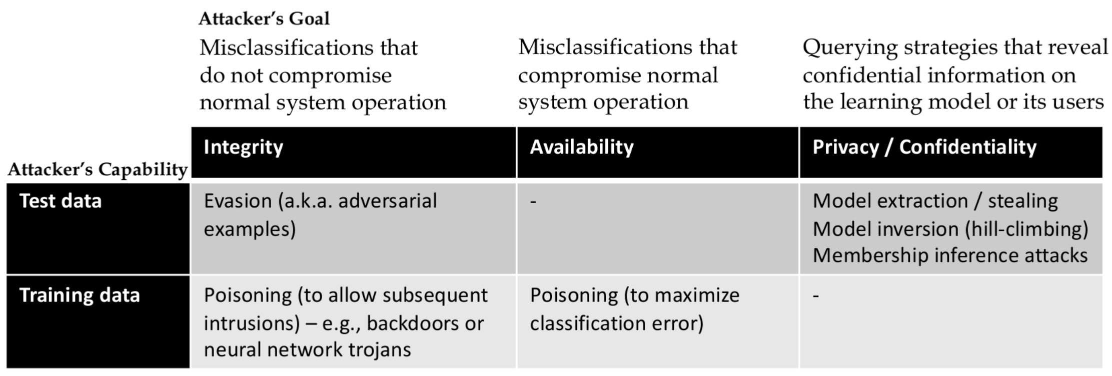
The adversary's capability specifies how much and which dataset it can modify. For example, the attacker can perform a causative attack that affects both training and the test set (poisoning attack) or an evasion attack that only affect the test set. The capability also includes, how many instances per dataset can be injected or modified, which instance classes, which features for each instance and how much. The ability can be expressed in terms of constraints, for example the maximum amount of changes that the attacker can make to an instance to prevent the attack from being perceived by the human eye.

2.2.4 Attack Strategy

The attack strategy identifies how the attacker should behave to maximize his goal, manipulating the instances of the training and/or test set, without violating the constraints of capability and knowledge. To obtain the optimal attack strategy, the attacker must solve an optimization problem, whose objective function is the adversary's goal and its constraints are given by the attacker's capability and his knowledge of the system.

2.3 Type of attack

This section summarizes the most common adversarial learning attacks and explains how they work. In [8] they introduced the Table 2.1 which collects the attacks that an adversary can do in relation to his capability and goal. Two very common



	Attacker's Goal		
	Misclassifications that do not compromise normal system operation	Misclassifications that compromise normal system operation	Querying strategies that reveal confidential information on the learning model or its users
Attacker's Capability	**Integrity**	**Availability**	**Privacy / Confidentiality**
Test data	Evasion (a.k.a. adversarial examples)	-	Model extraction / stealing Model inversion (hill-climbing) Membership inference attacks
Training data	Poisoning (to allow subsequent intrusions) – e.g., backdoors or neural network trojans	Poisoning (to maximize classification error)	-

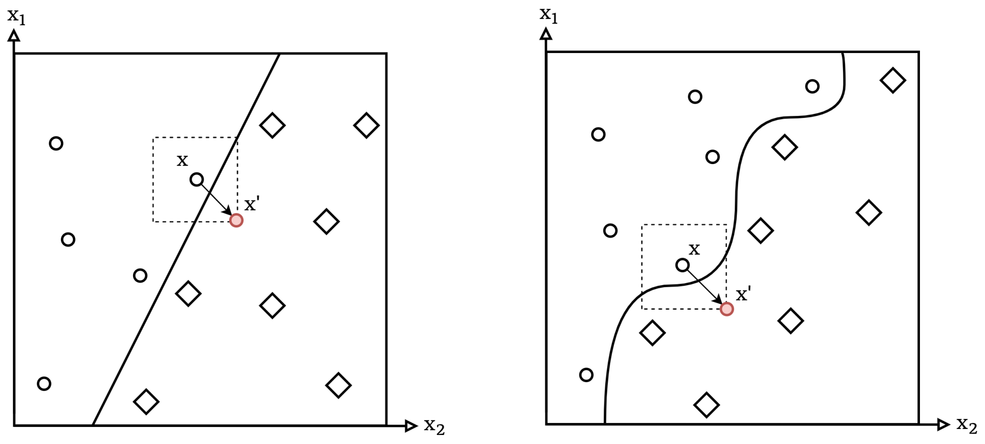
forms of attack are *Evasion Attack* and *Poisoning Attack*. The *Evasion Attack* is based on changing instances of the test set to cheat the target model and the adversary's capability is constrained by how much an instance can be changed. Instead a *Poisoning Attack* focuses on the training set instances with the aim of generating the highest prediction error. In this type of attack the adversary's capability is constrained by the maximum number of samples that can be added to the training set. Another type of attack is the *Model Inversion Attacks*, which try to extrapolate information from the training set by modifying an input instance of the model. These three types of attack are explained in detail in the following sections. Particular attention is given to evasion attacks as the robustness of the model proposed in this thesis is based on them.

2.3.1 Evasion attacks

An *evasion attack* occurs during the testing phase. Given an already trained model, the attacker alters the input instances to get an incorrect prediction from the model. These instances are called adversarial examples (or evading instances). In this type of attack the adversary can have a constraint on the number of features to attack (attackable features) and on how much he can perturb each feature. Each perturbation performed on an instance has a cost. The constraint of evasion attacks is modeled through a variable b called a budget. The attacker can modify the instance as long as he has enough budget. The adversary's goal is to find the perturbation that generates the highest prediction error while remaining within the contraints. Constraints are important in making a possible attack reasonable. In general a perturbation should not be perceptible to the human eye. This type of attacks works both on linear classifiers (NB, SVM linear, etc.) and on more complex models (NN, DT, etc.). For example given a linear model h and an instance $\boldsymbol{x} \in \mathbb{R}^d$ with output $y \in \mathbb{R}$. If $h(\boldsymbol{x}) < 0$ then to create an adversarial example $\boldsymbol{x}'$ such that $h(\boldsymbol{x}') > 0$, it is sufficient to modify the feature values $\boldsymbol{x}'$ along the direction of the decision boundary until the value of $h(\boldsymbol{x}')$ increases. The differences between $\boldsymbol{x}$ and $\boldsymbol{x}'$ must be subject to the budget constraint (the adversary's capability). The search for $\boldsymbol{x}'$ is solved through an optimization problem, both for linear and non linear models. In the case of a nonlinear model, the search for an adversarial example can be difficult as the decision boundary can be much more complex and not have a direct relationship between the features in $\boldsymbol{x}$ and the model parameters. In Figure 2.2 two adversarial examples are depicted in the context of a linear and a nonlinear model rispectively. To find an adversarial example in a nonlinear model, an attacker can use approximation techniques such as **Projected gradient descent** [37] which seeks an approximate solution for smooth functions. Evasion attacks can also be divided into **binary evasion attack** and **multiclass evasion attack**.

Binary evasion attack

In a binary evasion attacks, the attacker tries to push an instance as far as possible beyond the decision boundary that divides the instances. More formally, it is possible to generalize the research for an adversarial example for the model h



(a) Linear model (b) Non-linear model

Figure 2.2: The regions around the point $\boldsymbol{x}$ represent the constraint imposed by $\| \boldsymbol{x} - \boldsymbol{x}' \|_\infty \leq b$.

through the following optimization problem:

$$\begin{aligned} &\min_{\boldsymbol{x}' \in \mathbb{R}^d} h(\boldsymbol{x}')y \\ &s.t. \| \boldsymbol{x} - \boldsymbol{x}' \|_\rho \leq b \\ &\quad \boldsymbol{x}_{lb} \preceq \boldsymbol{x}' \preceq \boldsymbol{x}_{ub} \end{aligned} \tag{2.1}$$

where $u \preceq v$ means that each element of u has to be less or equal to the corresponding element in v. The smaller the $h(\boldsymbol{x}')y$ value, the larger the error produced by h in predicting $\boldsymbol{x}'$. The two constraints ensure consistency between $\boldsymbol{x}'$ and the capability of the adversary. The constraint $\| \boldsymbol{x} - \boldsymbol{x}' \|_\rho \leq b$ implies that the distance between $\boldsymbol{x}$ and $\boldsymbol{x}'$, with respect to the distance function L_ρ must be less than budget. The box constraint $\boldsymbol{x}_{lb} \preceq \boldsymbol{x}' \preceq \boldsymbol{x}_{ub}$ has two roles. For example, the first ensures that if the instance space $\boldsymbol{x}$ is bounded in $[0,1]^d$ then the instance $\boldsymbol{x}'$ will remain within this range by setting $\boldsymbol{x}_{lb} = \mathbf{0}$ and $\boldsymbol{x}_{ub} = \mathbf{1}$. Second, it can model the constraint on non-attackable features. If the features f is not attackable, it is sufficient set $x_{lb}^{(f)} = x_{ub}^{(f)} = x^{(f)}$ in the optimization problem.

Multiclass evasion attack

In a multiclass problem the model must discriminate an instance belonging to a specific class among many others. This brings errors on different classes. The attacker can take advantage of it and perform two types of attack [38]:

- **Error-generic evasion attacks** which attempt to misclassify an instance into any of the other classes.

- **Error-specific evasion attacks** which attempt to misclassify an instance into a specific class.

In particular in the error-generic scenario, the adversary has an interest in attacking the model, regardless of the class in which the adversarial example is misclassified. It is sufficient that the ending class is different from the original. To give a concrete example, a well-known criminal has an interest in not being recognized by a video surveillance system, but is not interested in the identity with which he is mistakenly associated. Instead in the error-specific scenario, the attacker wants to get a specific class as a model response. This can be seen as a person trying to authenticate himself as a specific user inside a system. In this case the attacker is interested in being misclassified as that person.
The search for an adversarial example for each of the two scenarios is formally defined below in the respective sections.

Error-generic evasion attack To generate an error-generic evasion instance, the attacker must solve the following optimization problem:

$$\begin{aligned} &\min_{\boldsymbol{x}' \in \mathbb{R}^d} \Delta(\boldsymbol{x}') \\ &s.t. \parallel \boldsymbol{x} - \boldsymbol{x}' \parallel_\rho \leq b \\ &\quad \boldsymbol{x}_{lb} \preceq \boldsymbol{x}' \preceq \boldsymbol{x}_{ub} \end{aligned} \tag{2.2}$$

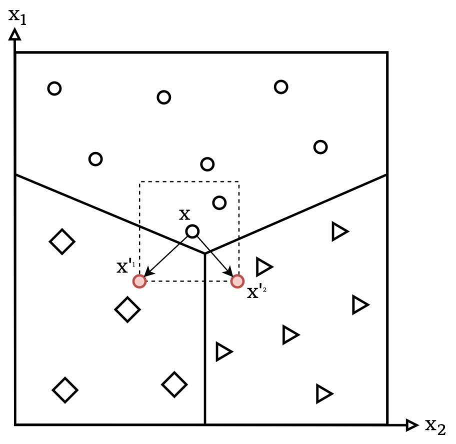
with $\Delta(\boldsymbol{x}')$ defined as:

$$\Delta(\boldsymbol{x}') = h_k(\boldsymbol{x}') - \max_{k \neq l} h_l(\boldsymbol{x}'). \tag{2.3}$$

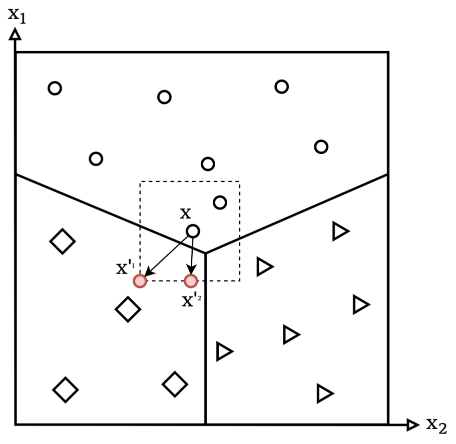
The k index represents the original class of the instance $\boldsymbol{x}$. Instead l represents any class, different from the original one, in which the attacker tries to misclassify the evading instance $\boldsymbol{x}'$. The maximization problem $\max_{k \neq l} h_l(\boldsymbol{x}')$ looks for the wrong class $l \neq k$, that has the highest score. The minimization of $\Delta(\boldsymbol{x}')$ represents the search for the perturbed instance $\boldsymbol{x}'$ such that the difference between the score in the class k and the highest one in another class l is the minimum. This means finding $\boldsymbol{x}'$ misclassified as the closest class. In Figure 2.3a it is possible to see an example of generic-evasion attack.

Error-specific evasion attack To generate an error-specific evasion instance, the attacker must solve the following optimization problem:

$$\begin{aligned} &\max_{\boldsymbol{x}' \in \mathbb{R}^d} \Delta(\boldsymbol{x}') \\ &s.t. \parallel \boldsymbol{x} - \boldsymbol{x}' \parallel_\rho \leq b \\ &\quad \boldsymbol{x}_{lb} \preceq \boldsymbol{x}' \preceq \boldsymbol{x}_{ub} \end{aligned} \tag{2.4}$$



(a) Error-generic evasion attack



(b) Error-specific evasion attack

Figure 2.3: In figure (a), the adversary looks for the closest class to the perturbations of $\boldsymbol{x}$. Instead in figure (b) the adversary looks for the perturbation of $\boldsymbol{x}$ which has the highest score in the target class.

The difference with the error-generic scenario lies only in the change of the minimization problem into a maximization problem. The k index represents the adversary's target class, the one in which he wants his evading instance to be misclassified. The l index, on the other hand, represents the starting class of the instance $\boldsymbol{x}$. In this case the attacker looks for the perturbation of $\boldsymbol{x}$ which generates the highest score difference between the target class k and the original class l. The bigger the difference (maximization problem), the bigger the model error when misclassify $\boldsymbol{x}'$ as k instead of l. In Figure 2.3b it is possible to see an example of specific-evasion attack.

2.3.2 Poisoning attacks

In a *poisoning attack* the adversary's goal is to maximize the classification error through the injection of poisoning samples into training set. In this type of attacks the adversary's capability is given by the maximum number of samples it can add to the training set (a small part). To perform this type of attack, the adversary must have perfect knowledge of the model (white box attack). The maximization of the classification error is done by finding an optimal attack point $\boldsymbol{x}_c$ through the following optimization problem [39]:

$$\begin{aligned} &\max_{\boldsymbol{x}_c \in \mathbb{R}^d} \mathcal{L}_{val}(\mathcal{D}_{val}, \hat{h}) \\ &s.t.\ \hat{h} = \arg\min_{h \in \mathcal{H}} \mathcal{L}_{tr}(\mathcal{D}_{tr} \cup \{(\boldsymbol{x}_c, y_c)\}, h). \end{aligned} \tag{2.5}$$

The attacker looks for the instance $\boldsymbol{x}_c$ that force the generation of a model that has the lowest loss in the training set $\mathcal{D}_{tr} \cup \{(\boldsymbol{x}_c, y_c)\}$ and the highest loss (maximize generalization error) in classification of untainted data (validation set $\mathcal{D}_{val}$). The attacker somehow infects the training set with malicious instances, with the aim of training the model incorrectly, so as to compromise its performance during the test phase. If you wonder how the attacker can infect a model's training set, just think of the fact that machine learning algorithms are often re-trained on data that they collect over time. This is done to adapt the models to the changes in the underlying data distribution. This happens in intrusion detection systems, which can be re-trained on a sample of data collected during system execution. An attacker can take advantage of this re-training technique to injecting carefully designed samples to eventually compromise the whole learning process. Below we give a definition of a specific poisoning attack, the *backdoor attack* (also known as poisoning integrity attack).

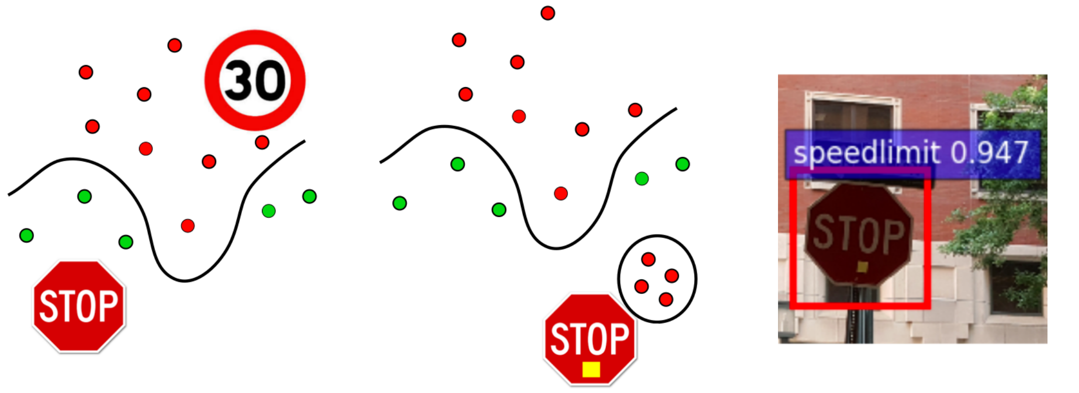
Backdoor attacks / Poisoning integrity attacks In a backdoor attack, the adversary manipulates a pre-trained network models to create specific backdoor vulnerabilities. The attacker injects mislabeled samples into a feature space region far from the original training instances. In this way the learning algorithm labels such region as desired and generates vulnerable points leading to intrusions or misclassifications at test time. Subsequently, he publicly releases the corrupt model so that it can be used in most proprietary systems. Once the attacker encounters one of these systems, he can activate the backdoot using specific input samples that are misclassified as desired [8]. Figure 2.4 gives an example of a possible poisoning attack applied to the recognition of traffic signs.

2.3.3 Model inversion attacks

Model inversion attack is an attack that undermines the privacy of information used by the model to make predictions. In this type of attack, the adversary interrogates the model with a fictitious instance. He repeatedly interrogates the model and modifies the input instance in order to maximize the model's output score. In this way the attacker can reconstruct a sample used during the training of the model quite accurately. For example, in [22] they show how in a context of facial recognition, given the name of a user in the system, the attacker can extrapolate the image of the user's face used during his registration in the system.

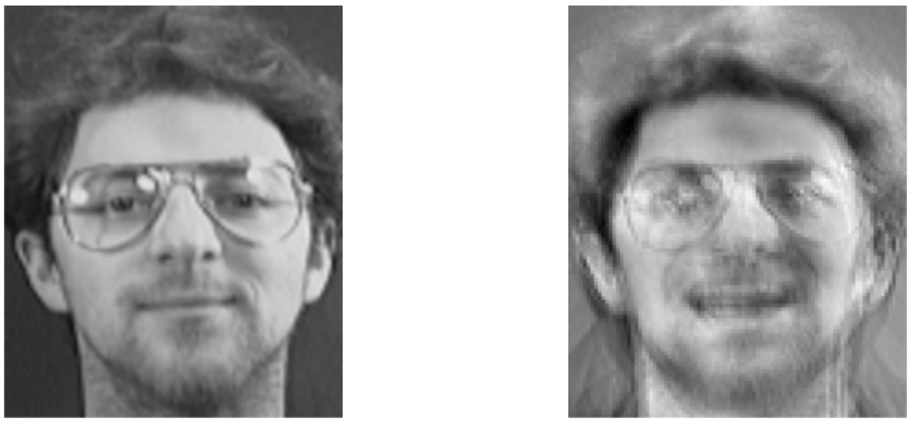
2.3.4 Transferability attacks

A reason why a gray/black box context does not guarantee adversarial attack robustness can be found from one of the properties that adversarial examples have.



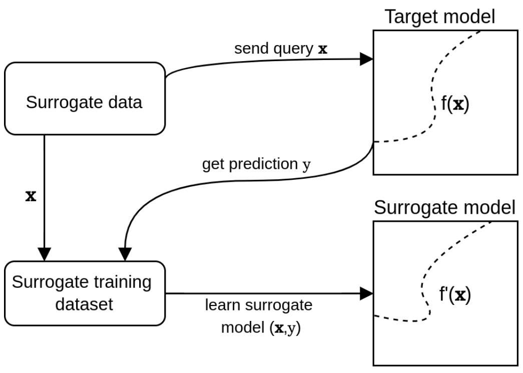
(a) Normal training data (b) Poisoned training data (c) Backdoor in test data

Figure 2.4: Figure (a) shows the decision boundary of a model trained with legitimate instances. In figure (b) the attacker has poisoned the training set by inserting some images of a perturbed stop sign with a yellow square labeled as a speed limit. In figure (c) it can be seen how a real word image of a stop signal perturbed in a similar way to poisoning instances activates the backdoor inserted by the attacker. The perturbed stop signal is misclassified as a speed limit with great confidence. This simple example highlights the possible disastrous consequences of this type of attack in a context of self-driving cars. This example was taken from [8].

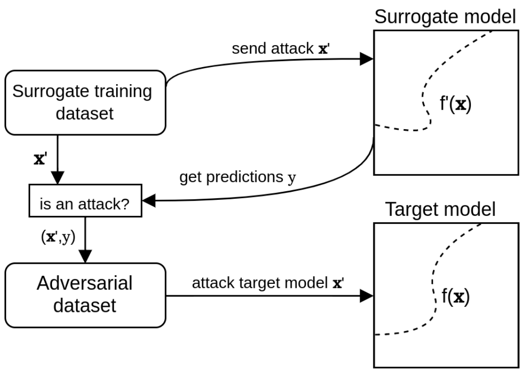


(a) Origninal training sample (b) Retrieved training sample

Figure 2.5: In figure (a) you can see the original image used in the training set. Instead figure (b) is the image retrieved by the attacker using a model inversion attack. In this experiment, the attacker only knew the user's name and had the opportunity to interrogate the facial recognition system to obtain a class confidence score. Images taken from [22].



(a) Surrogate model train



(b) Transferability attack

Figure 2.6: Figure (a) shows the process of creating the surrogate model, by interrogating the target model. While in figure (b) it is shown how a transferability attack is performed on the target model through adversarial examples generated on the surrogate model.

It has been noticed that different models trained on the same (or similar) trainig set are attackable by the same perturbations to the input instances. From this discovery attacks by transferability were born. This type of attack is generated on a surrogate model and then used to attack the target model. This type of attack is widely used where the generation of attacks for a given model is difficult, due to the complexity of the model, or in a context of gray/black box attack, where there is a lack of knowledge of the target model. For example, suppose a limited-knowledge context, where the attacker has some surrogate data sampled from the same data distribution of the training set, he knows the features representation and the learninig algorithm, and he has the possibility to interrogate the target model. With this information the attacker can build a model close to the one to attack and generate attacks from it. To build this model, the attacker first interrogates the target model with the surrogate instances and assigns to them the its prediction. In this way the attacker creates a dataset similar to the one used for training the target model. Once the surrogate training dataset is obtained, the attacker trains a surrogate model with the same learning algorithm used by the model to attack. Once the model is created, the attacker generates attacks on it and then uses transferability to attack the target model. This type of attack is also called substitute models and black-box attack. The procedure for creating a substitute models attack is shown in Figure 2.6.

2.4 Countermeasures

This section briefly summarizes the use practices and strategies used to improve the robustness against the attacks exposed in the previous sections. More specific and detailed cases are reported in the following chapter of the **state of the art** of robust models 3.
In [1] they propose the Table 2.2 which summarizes the techniques: *regularization*, *randomization* and *information hiding*, to defend a model against the attacks collected in the Table 2.1.

Table 2.2: Countermeasures against adversarial machine learning. T and I are rispectively Targeted and Indiscriminate attack. Part of the information in this table was taken from [1]

		Integrity	Availability	Privacy
Causative	T	• Regularization • Randomization	• Regularization • Randomization	• Regularization • Randomization
	I	• Regularization	• Regularization	• Regularization
Exploratory	T	• Information hiding • Randomization	• Information hiding	• Information hiding • Randomization
	I	• Information hiding		• Information hiding

Regularization The technique called regularization allows to increase the robustness against *causative* attacks by adding a constraint in the search for the best hypothesis $\hat{h}$ during training. The problem 1.1 of finding the best learner $\hat{h}$ proposed in the supervided learning section 1.1 is modified in the following way:

$$\hat{h} = \arg\min_{h \in \mathcal{H}} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} \ell(y, h(\boldsymbol{x})) + \lambda\Omega(h) \tag{2.6}$$

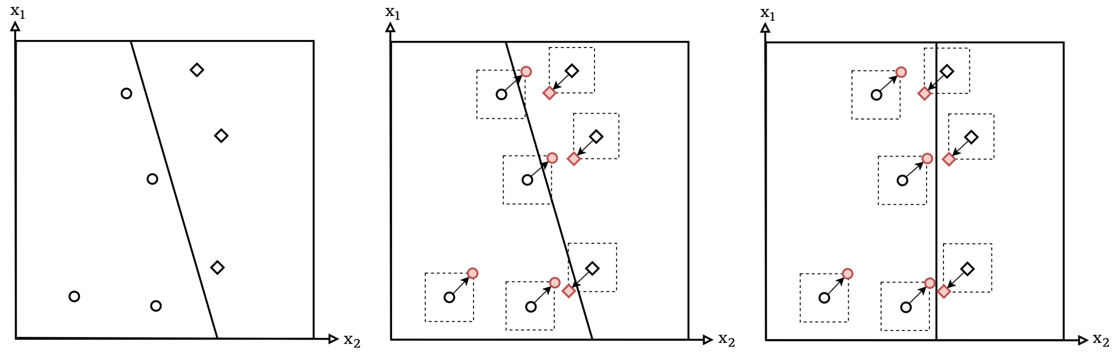
where the loss function $\mathcal{L}$ of 1.1 is replaced with the aggregation of the *instance-level losses*. The value $\Omega(h)$ is the penalty term and the parameter λ is the regularization term used to manage the trade-off between precision of the model and robustness introduced by the penalty term. Regularization is used to restrict or bias the choice of h when there is a low amount of data or the data has noise. Regularization can also be interpreted as prior encoding of the parameters, penalizing the choice of those that are less likely a priori. It is important to note that there is a strong relationship between constraint and expressivity. The more a model includes prior information, the more it loses flexibility in adapting to data, but it becomes more robust. Instead the more expressive the model becomes, the more it includes information extrapolated from the data and the more vulnerable it becomes to attacks. In other words, the use of prior information (or constraints) generates a model that depends less on the fit on the data and therefore less attack opportunities are possible for the adversary. The λ parameter managed the trade-off between the two cases.

Randomization Randomization generates a more robust model against targeted attacks. In fact, unlike indiscriminative attacks, targeted attacks are more sensitive to changes in the decision boundary. Even a small movement of the decision boundary can change the classification of the relevant points. For this reason, the randomization technique introduces randomness in the placement of the decision boundary. In this way the adversary receives imperfect feedback from the learner and is forced to make a greater effort to reach the target class. The disadvantage of randomization is that increase the error-rate at test time.

Information hiding The information hiding is a practice that tries to keep the model information hidden to the attacker in order to put it in front of a black-box scenario. However even if the adversary does not have knowledge of the learning algorithm, it cannot be excluded that he may somehow infer some information on the training set. The Kerckhoffs' principle states that the security of a system should not rely on unrealistic expectations of secrecy. In designing a robust model, only minimal assumptions should be made about what must realistically remain a secret. In general it is better to assume that the atacker always knows which is the learning algorithm and he can obtain information about the training set.

2.4.1 Countering Evasion

To defend against evasion attacks there are generally two strategies. The first is called *robust optimization*, which reduces the sensitivity of the robust model in input changes (small perturbations do not change the prediction of the model). The second one is called *rejection/detection*, which limits the input space considered as admissible.



(a) Normal training (b) Model under attack (c) Adversarial training

Figure 2.7: The example shows the effects of adversarial learning on the robustness of the trained model. Figure (a) shows a possible decision boundary that perfectly divides the traing data. In figure (b) it can be seen that an attacker can generate three adversarial examples. Figure (c) shows how adversarial treaning finds a more robust decision boundary with less errors under attack.

Robust optimization

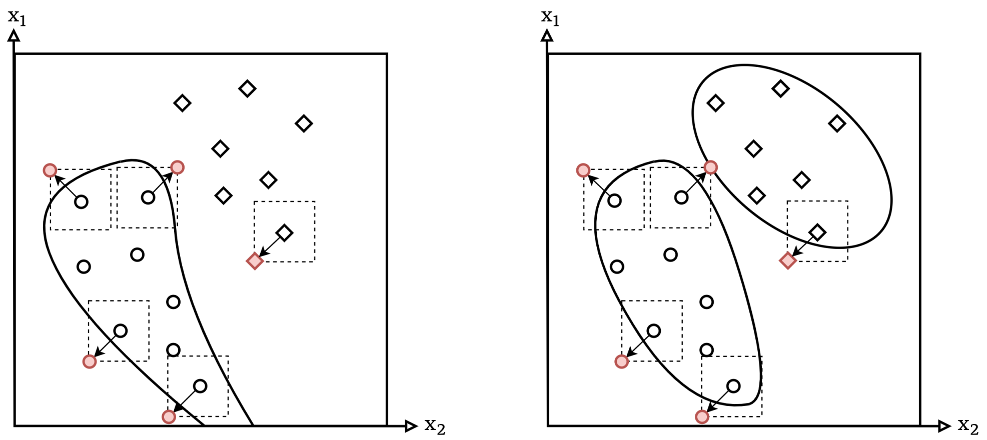
Robust optimization is further divided into regularization (described above) and adversarial training. **Adversarial training** is the practice of training a robust model using perturbed instances based on the adversary's capability together with the normal training set.

$$\hat{h} = \arg\min_{h \in \mathcal{H}} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} \max_{\|\delta\|_\rho \leq b} \ell(y, h(\boldsymbol{x}+\delta)) \tag{2.7}$$

where $\| \delta \|_\rho \leq b$ is the perturbation generated with respect to the distance function L_ρ and $\boldsymbol{x}+\delta$ is the evading instance used in the training of h. In this way the learning algorithm does not find $\hat{h}$ which minimizes the loss on normal instances as in 1.1 but looks for the hypothesis $\hat{h}$ which minimizes the maximum loss generated by possible attacks with respect to the adversary's capablity. An example of adversarial training is shown in Figure 2.7. The possible attacks for the model in Figure 2.7b are used to train the robust model in Figure 2.7c. The perturbed instances used in the train are not present in the normal data distribution, therefore the attacker cannot generate a new attack starting from those, but can only generate attacks starting from the instances in the normal distribution.

Detecting / Rejecting

Very often, the adversarial examples tend to occur in blind spots, regions of space far from the training points. In this type of attack, adversarial examples do not



(a) Normal training (b) Detection / Rejection training

Figure 2.8: The samples of the *circle* class which were previously classified as *diamonds*, figure (a), are now rejected by the mode, figure (b).

need to resemble instances in the target class. It is enough that they are beyond the decision boundary of the original class. As can be seen from the example in Figure 2.8a, many points of the *circle* class can be pushed out of the region and be misclassified as *diamond*. Through the use of detection/rejection technique it is possible to enclose the training points of each class in a class region and not to consider admissible all the instances that are predicted outside a region. As shown in Figure 2.8b, all *circles* classified as *diamonds* are now rejected (and vice-versa).

2.4.2 Countering Poisoning

As explained above, in a poisoning attack, the adversary inserts one or more outliers into the training set in order to make the learning algorithm produce a model with poor performance at test time. To defend against this type of attack it is possible to use two strategies, *data sanitization* and *robust learning*.

Bagging To perform data sanitization it is possible to use Bagging 1.3.1 on training instances to remove poisoning samples. As explained in [2] the poisoning samples are often outliers and Bagging in fact equalizes the influence of training samples, reducing the influence of the outlier in training data [26]. There is also a weighted Bagging alternative that resample the training set by assigning a probability distribution over training samples, in particular, lower probability weights to the most outlying observations.

Reject-On-Negative-Impact (RONI) Reject-On-Negative-Impact defense is another data sanitization strategy that identifies posoning attacks. For each samples in the training, the algorithm measures the pefromances of the model trained with or without the sample [40]. Through the average change in performance between the two models, the algorithm measures the impact that each sample has. Samples with negative influence on model performance are eliminated from the training set.

TRIM TRIM [31] is an iterative robust learning algorithm that trains a robust linear regression model that has poisoned training set. A linear regression model is defined as $h(\boldsymbol{x}) = \boldsymbol{w}^T + b$, with y being the prediction for the sample $\boldsymbol{x}$. The model h is parameterized by the vector $\theta = (\boldsymbol{w}, b) \in \mathbb{R}^{d+1}$. Where $\boldsymbol{w} \in \mathbb{R}^d$ is the vector of the feature weights and the parameter $b \in \mathbb{R}$ is the bias. For each iteration i, the TRIM algorithm uses a trimmed loss function to sample a new subset $\mathcal{D}_i \subset \mathcal{D}$ of size n (number of sample safe in the dataset) which minimizes the loss $\mathcal{L}(\mathcal{D}_i, \theta_{i-1})$. The subset $\mathcal{D}_i$ have the points with lowest residuals and therefore poisoned instances are excluded from the training. To find the subset of size n with lowest residuals, the algorithm solves the following optimization problem:

$$\begin{aligned} &\mathcal{D}_i = \underset{\mathcal{D}' \subset \mathcal{D}}{\arg\min}\, \mathcal{L}(\mathcal{D}', \theta_{i-1}) \\ &s.t.\ |\mathcal{D}'| = n \end{aligned} \tag{2.8}$$

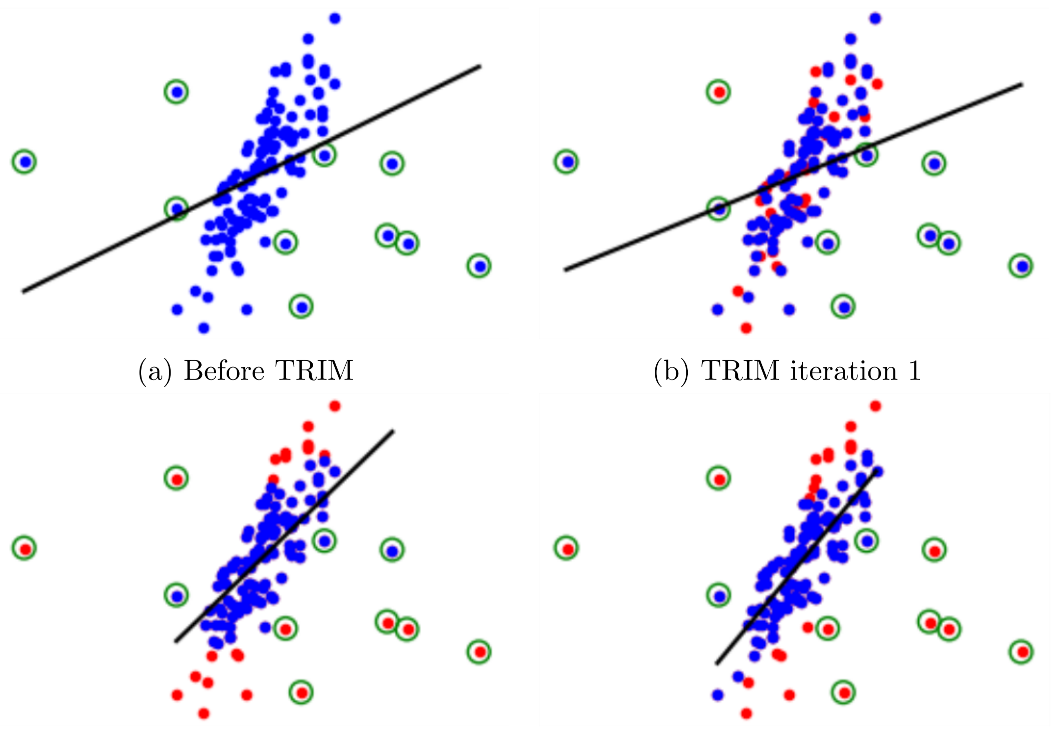
Initially $\mathcal{D}_0$ is generated by randomly taking n samples from $\mathcal{D}$. The samples in $\mathcal{D}_i$ are the training points with lowest residuals for the regression model h with parameters θ_{i-1}. Obviously this subset of point can also contain poisoned samples, but they are similar to legitimate samples and do not contribute much to poisoning the model. From this subset the new parameters θ_i are calculated through the following minimization problem:

$$\theta_i = \underset{\theta \in \mathbb{R}^{d+1}}{\arg\min}\, \mathcal{L}(\mathcal{D}_i, \theta) \tag{2.9}$$

where $\mathcal{L}(\mathcal{D}_i, \theta)$ is:

$$\mathcal{L}(\mathcal{D}_i, \theta) = \underbrace{\frac{1}{n} \sum_{(\boldsymbol{x},y) \in \mathcal{D}_{i-1}} (h(\boldsymbol{x}, \theta) - y)^2}_{MSE(\mathcal{D}_i, \theta)} - \lambda \Omega(\boldsymbol{w}) \tag{2.10}$$

where the Mean Squared Error $MSE(\mathcal{D}^i, \theta)$ measures the error in predictions made by $h(\mathcal{D}_i, \theta)$. $\Omega(\boldsymbol{w})$ is a regularization term which penalizes large weight values, and λ is called regularization parameter, as for regularization 2.6. The algorithm ends when the estimation of the parameters θ converges and the loss function reaches the minimum. Figure 2.9 shows the effect of the TRIM algorithm at each iteration.



(a) Before TRIM (b) TRIM iteration 1

(c) TRIM iteration 2 (d) TRIM iteration 3

Figure 2.9: The blue points are those belonging to the training set. At each iteration, the poisoning samples (the outliers) are excluded from the training set (red points). The example was taken from [31].

Summary

In this chapter we have addressed one of the main themes of this thesis, the adversarial machine learning. We have given the definition of attack and adversary, and we have shown various types of attacks and countermeasures. We have summarized the contents of each section of the chapter below:

- **Attack taxonomy.** In short, we gave the definition of attack based on three fundamental features: **influence** which specifies the intention of the attack to cause damage or extract information from the model, **security violation** that states the type of violation performed (integrity, availability or privacy) and **specifity** that identifies whether the attack is targeted or indiscriminate.

- **Adversary's Model.** We have given the definition of adversary (attacker), modeled on four fundamental points: the **adversary's goal** that specifies the purpose of the attack, the **adversary's knowledge** with respect to the target model (white, gray, black box), the **adversary's capability** which defines what the attacker can modify (training and/or test sets) and how much it can perturb each sample, and finally the **attack strategy** that identifies how the attacker must behave to achieve his goal while maintaining all the constraints.

- **Type of attack.** In this section we have summarized the most common types of attacks for supervised learning models with particular attention to **evasion attacks** (error-generic evasion attacks and error-specific evasion attacks), **poisoning attack** (backdoor attacks), **model inversion attacks** and **transefability attacks**. We have given an example for each attack.

- **Countermeasures.** Finally in the last section of this chapter we gave a general introduction on the countermeasures to be adopted to design a robust model such as detecting/rejectiong, robust optimization (**adversarial training**) for evasion attacks and RONI, TRIM and Bagging for poisoning attacks.

Chapter 3

State of the art

This chapter deals with the current state of the art of finding robust models against adversarial machine learning attacks. The first part of this section briefly summarizes some works on the state of the art on countermeasures against adversarial machine learning attacks of the most common models. Given the nature of this thesis, in the second part particular attention is given to research relating to decision trees. Furthermore, this chapter is important for understanding the difference between the proposed work and what has already been done.

3.1 Countermeasures for SVM, NN

Since it was discovered that machine learning models are vulnerable to various types of attacks, researchers in the field of machine learning have started to design robust models, both for linear and non-linear models. For example, SVM is one of the best known machine learning models and in [4] they have shown how it is subject to evasion, poisoning and privacy attacks. In the same article they show for each of these attacks a possible countermeasure through an adversary-aware design of SVMs. In [15] instead they developed a version of SVM called Randomized-SVM robust against generalized adversarial attacks under uncertanty. R-SVM trains a distribution of classifier instead of one like the classic SVM. In [25] they investigate the vulnerability of neural networks to adversarial examples, stating that the cause of this vulnerability lies in the linear nature of the NN. Successively they created a fast method for generating adversarial examples against NN, which were used to train a robust maxout network with adversarial training. In [34] they proposed a way to scale adversarial training efficiently to large models and datasets and showed how to solve the label leaking effect that leads adversarially trained models to be more robust on perturbed instances than on original instances. In [37] they studied the robustness of a neural network using robust optimization through a natural saddle point formulation to capture the notion of security against adversarial attacks. In [11] they extend the work proposed by [37] on adverarial training. They say that a robust model must be created gradually with increasingly stronger attacks and not immediately with the stronger attack. To do this they presented the *Curriculum adversarial training* an optimized adversarial training approach. In addition, to prevent the model from forgetting the weakest attacks (*catastrophic forgetting*), during the training phase they use the *batch mixing* technique which introduces lower intensity attacks during the training of stronger attacks. Finally, to counter the *attack generalization* problem, they use the *quantization* technique that reduces the space of adversarial examples. This is just a small overview of research in adversarial machine learning. New attacks are continually being developed to evade robust models and new models are designed to be robust against these attacks.

3.2 Boosted Decision Tree - Robust

In [32] two new algorithms are proposed to create evasion attack for tree ensembles such as boosted trees and random forests. In addition they presented a new robust model based on Boosted Decision Tree (BDT).
The first algorithm is based on **Mixed Integer Linear Program** (**MILP**), which finds the optimal solution at a high computational cost. The second algorithm is called *symbolic prediction* and sacrifices optimality in favor of speed, but still generating good evading instances. Through these algorithms, they empirically demonstrated that both Random Forest and Boosted Trees are weak compared to evasion attacks. These two algorithms are covered in more detail in the sections **Optimal evasion** 3.2.1 and **Approximate evasion** 3.2.2.
The robust model they proposed is called **BDT-R**, and is based on boosted tree. At each boosting iteration, the model is trained with the evading instances produced by symbolic prediction. This learning technique has been called *adversarial boosting*. The creation of this model is explained in detail in the **BDT-R model** 3.2.3 section.

The concept behind an evasion instance can be summarized in the following lines. Let $\mathcal{X} \subseteq \mathbb{R}^d$ be the set of all possible instances, $\mathcal{Y} = \{-1, 1\}$ the label set and $h : \mathcal{X} \to \mathcal{Y}$ a classifier. For an instance $\boldsymbol{x} \in \mathcal{X}$ and given a distance function $dst : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^+$, the optimal evasion problem is defined as:

$$\min_{\boldsymbol{x}' \in \mathcal{X}} dst(\boldsymbol{x}, \boldsymbol{x}') \qquad \text{subject to } h(\boldsymbol{x}) \neq h(\boldsymbol{x}'). \tag{3.1}$$

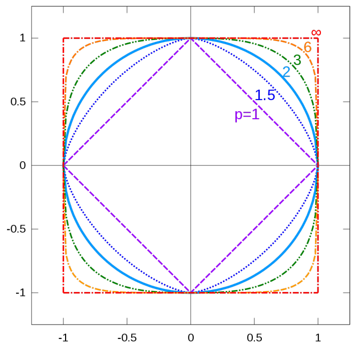
This means finding the instance $\boldsymbol{x}'$ with the minimum distance $dst(\boldsymbol{x}, \boldsymbol{x}')$, which is classified differently from $\boldsymbol{x}$. The optimal evasion algorithm was modeled to work

Table 3.1: Useful definitions to understand the reduction of the optimization problem. 3.1

Notation	Definition
$\mathcal{T}$	trees ensemble
t	a tree inside the ensemble: $t \in \mathcal{T}$
$t.nodes$	the set of tree nodes
$t.leaves$	the set of tree leaves
σ	a generic node or leaf

with the distance functions L_0, L_1, L_2 and L_∞, while the approximate version only for L_0. These distances are defined as follows (the figure show the geometric shape of the most common norms). The evasion attack algorithms proposed in [32] have

- The L_0 distance: $\sum_{f=1}^{d} \mathbb{I}_{x^{(f)} \neq x'^{(f)}}$
- The L_1 distance: $\sum_{f=1}^{d} |x^{(f)} - x'^{(f)}|$
- The L_2 distance: $\sqrt{\sum_{i=1}^{d} (x^{(f)} - x'^{(f)})^2}$
- The L_∞ distance: $\max_f |x^{(f)} - x'^{(f)}|$



been modeled on an ensemble $\mathcal{T} : \mathbb{R}^d \rightarrow \mathbb{R}$, whose weak-learners are regression trees $t \in \mathcal{T}$. Each t is a binary regression tree, whose prediction is given by the value of the leaves. The ensemble's prediction $\mathcal{T}(\boldsymbol{x})$ is given by the sum of the predictions of the individual trees. If the threshold is zero, the label assigned to $\boldsymbol{x}$ will be $h(\boldsymbol{x}) = 1 \iff \mathcal{T}(\boldsymbol{x}) > 0$. For a tree ensemble $\mathcal{T}$, find an instance $\boldsymbol{x} \in \mathbb{R}^d$ such that $\mathcal{T}(\boldsymbol{x}) > 0$ (or $\mathcal{T}(\boldsymbol{x}) < 0$) is NP-Hard, regardless of the choice of dst.

3.2.1 Optimal evasion

The first algorithm is called *optimal evasion* and solves the 3.1 problem by reducing it to a MILP. The algorithm solves the problem in an optimal way finding the best evading instace. The shortcoming of this algorithm is that for very complex models, the computational complexity of finding the best solution is NP-Hard. The reduction occurs through the introduction of three groups of MILP variables (**predicate variables**, **leaf variables** and **objective variable**) and three families of constraints (**predicates consistency constraints**, **leaves consistency**

constraints and **model mislabel constraint**). To complete the reduction it is necessary to transform the **objective function** of 3.1 in order to relate the predicate variables to the value of $dst(\boldsymbol{x}, \boldsymbol{x}')$.

Variables The MILP variables are defined as follows:

- The **predicate variables** are binary variables $p_i \in \{0, 1\}$, of number at most $\sum_{t \in \mathcal{T}} |t.nodes|$. Given a instance $\boldsymbol{x}$ to be perturbed, each variable p_i represents the state of a predicate of a node with (f, v) as spliting point, such that $p_i = 1 \iff x^{(f)} \leq v$, otherwise $p_i = 0$.
- The **leaf variables** are continuous variables $0 \leq l_i \leq 1$, of number at most $\sum_{t \in \mathcal{T}} |t.leaves|$. The MILP constraints force exactly one non-zero l_i per tree, with $l_i = 1$. The leaf variables indicate which prediction leaf is active in each tree.
- The **objective variable** is a non-negative continuous variable denoted as b (bound). This variable is used to express the distance $dst(\boldsymbol{x}, \boldsymbol{x}')$ of 3.1, when the distance function L_∞ is used.

Constraints The family of constraints is divided into three parts, each of which maintains consistency between the variables defined above. Following, the three types of constraints are introduced.

Predicates consistency constraints The predicates consistency constraints are used to ensure that the predicate variables are logically consistent with each other. Each predicate variable p_i is associated with a predicate state $x^{(f)} \leq v_i$. If p_i and p_j are predicate variables on the same feature f such that $x^{(f)} \leq v_i$ and $x^{(f)} \leq v_j$, it is possible that p_i and p_j take on inconsistent values without the aid of further constraints. For example, if $v_i \leq v_j$, it is not possible to have $p_i = 1$ and $p_j = 0$. For each feature variable $x'^{(f)}$, the consistency of its predicates p_i is ensured by introducing $K - 1$ inequalities, with K the number of predicates on $x^{(f)}$. In particular for $x'^{(f)}$, let $v_1 < \cdots < v_K$ be the ordered thresholds of its predicates and $p_1, \ldots, p_K$ be its predicate variables such that $x'^{(f)} < v_1, \ldots, x'^{(f)} < v_K$, a valuation of $(p_i)_{i=1,\ldots,K}$ is consistent if and only if $p_1 = 1 \Rightarrow \cdots \Rightarrow p_K = 1$. In the MILP reduction, the predicates consistency constraints are represented as:

$$p_1 \leq \cdots \leq p_K \tag{3.2}$$

Leaves consistency constraints Each tree has its own set of leaves consistency constraints between p and l. These constraints guarantee the following three properties, which are fundamental for maintaining the semantics between p and l.

1 If there is a leaf $l_k = 1$, then all other leaves $l_{i \neq k}$ must be equal to zero.

2 If a leaf $l_k = 1$, then all predicate variables p_i in the path from root to leaf l_k must be 0 or 1 according to the semantics of the prediction path of l_k.

3 A single leaf variable l_k per tree can be equal to 1.

Property 1 can be easily guaranteed through the following constraint:

$$l_1 + \cdots + l_K = 1 \tag{3.3}$$

with K the number of leaves in the tree. To guarantee property 2, two constraints are needed for each node. There are two cases, one for the root node and one for the internal nodes. The root predicate variable p_{root} is true if and only if the active prediction leaf belongs to the leaf variables of the left branch of the root, $l_1^T, \ldots, l_i^T$, while it is false if the active prediction leaf belongs to the leaf variables in the right branch, $l_1^F, \ldots, l_j^F$. Since only one leaf variable can be non-zero, constraints are written as:

$$1 - (l_1^F + \ldots, +l_j^F) = p_{root} = l_1^T + \ldots, +l_i^T \tag{3.4}$$

For internal nodes, the predicate variable p_{node} may never be active. This can happen if the active prediction leaf is not in any of the leaf variables below the node. These constraints are written as.

$$1 - (l_1^F + \ldots, +l_j^F) \geq p_{node} \geq l_1^T + \ldots, +l_i^T \tag{3.5}$$

Property 3 is automatically guaranteed through the two previous constraints.

Model mislabel constraint The model mislabel constraint is used to force the class difference between $\boldsymbol{x}$ and $\boldsymbol{x}'$. Let $\boldsymbol{x}$ be an original dataset instance, such that $\mathcal{T}(\boldsymbol{x}) < 0$, then to get an evasion instance, the perturbed instance $\boldsymbol{x}'$ must produce $\mathcal{T}(\boldsymbol{x}') \geq 0$. To encode the output of $\mathcal{T}(\boldsymbol{x}')$ given the leaf variables l for each tree, the following weighted sum must be performed.

$$\sum_{t \in \mathcal{T}} \sum_i o_i^t l_i^t \geq 0, \tag{3.6}$$

where o_i^t is the prediction value of the leaf i with respect to the regression tree t.

Objective function To complete the reduction of 3.1 in a MILP, the objective function $dst(\boldsymbol{x}, \boldsymbol{x}')$ must be translated as a function of p. For any distance L_ρ with $\rho \in \mathbb{N}$, there are a set of weights w_i and a constant C, such that the MILP objective function can be written as:

$$\sum_i w_i p_i + C. \tag{3.7}$$

This is possible because predicates discretize feature values and therefore the optimal distance $dst(\boldsymbol{x}, \boldsymbol{x}')$ can only take a finite number of values. If the distance L_∞ is used, then the objective function is represented by the variable b with d bounding constraints, each of which verifies that $|x^{(f)} - x'^{(f)}| \leq b, \forall f \in [1, d]$.

At this point, having defined all the variables and all the constraints, the complete reduction of 3.1 to MILP is denoted as:

$$\begin{aligned} \min_{p,l} \ & \text{objective function} \\ \text{s.t. } & \text{predicates consistency constraints} \\ & \text{leaves consistency constraints} \\ & \text{model mislabel constraint} \\ & \text{objective variable (bounding constraints when } dst = L_\infty) \end{aligned} \tag{3.8}$$

3.2.2 Approximate Evasion

The reduction of 3.1 in a MILP proposed in 3.2.1 can have a very significant solving time for difficult models. For this reason, the authors of [32] proposed an approximation of the evasion algorithm which allows to generate good quality evading instances with exponentially lower computational complexity. The approximation is based on the minimization of the distance function L_0. Given the instance $\boldsymbol{x}$ with $\mathcal{T}(\boldsymbol{x}) < 0$, the algorithm proposed looks for the $\boldsymbol{x}'$ instance that maximizes:

$$\boldsymbol{x}' = \underset{\tilde{\boldsymbol{x}}: \|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|_0 = 1}{\arg\max} \ \mathcal{T}(\tilde{\boldsymbol{x}}). \tag{3.9}$$

The instances $\boldsymbol{x}'$ and $\boldsymbol{x}$ differ by one feature. The approximation is called *symbolic prediction*, a dynamic programming approach that visits each node of the tree at most once. The symbolic instance $\tilde{\boldsymbol{x}}$ is moved from the root to the leaves of the tree taking into account the constraints imposed on it. if the instance cannot continue its descent since it would require the modification of two or more features then the exploration is terminated. At the moment a leaf is reached, if the instance $\tilde{\boldsymbol{x}}$ is different from $\boldsymbol{x}$, the *dimension-interval-prediciton* tuple is created. In algorithm 3 the symbolic prediction pseudocode proposed in [32] is reported. The `Symbolic Instance` s is a data structure to track constraints on $\tilde{\boldsymbol{x}}$, that it is defined with the following four methods:

- `isFeasible(`p`)`: return true if and only if there is an instance $\tilde{\boldsymbol{x}}$ such that $\| \tilde{\boldsymbol{x}} - \boldsymbol{x} \|_0 \leq 1$. This means that $\boldsymbol{x}'$ differs from $\boldsymbol{x}$ at most one feature, and all constraints including p hold.
- `Update(`p`)`: update the constraints on $\tilde{\boldsymbol{x}}$ by adding the predicate p.

Algorithm 3 SymbolicPrediction

```
Require: node $\sigma$ (either internal or leaf), the SymbolicInstace $s$ and the input/output set $u$.
 1: function SymbolicPrediction($\sigma, s, u$)
 2:     if $\sigma$ is a leaf then
 3:         if $s$.isChanged() then
 4:             $u \leftarrow u \cup \{s.\texttt{getPerturbation}(), \sigma.prediction\}$
 5:         end if
 6:     else
 7:         if $s$.isFeasible($\sigma.predicate$) then
 8:             $s_T \leftarrow$ copy($s$)
 9:             $s_T$.Update($\sigma.predicate$)
10:             SymbolicPrediction($\sigma.true, s_T, u$)
11:         end if
12:         if $s$.isFeasible($\neg\sigma.predicate$) then
13:             $s$.Update($\neg\sigma.predicate$)
14:             SymbolicPrediction($\sigma.false, s, u$)
15:         end if
16:     end if
17: end function
```

- `isChanged()`: returns true if and only if the current set of constraints imply $\boldsymbol{x} \neq \tilde{\boldsymbol{x}}$.
- `getPerturbation()`: return the tuple *dimension-interval*, with *dimension* be the feature f such that $x^{(f)} \neq \tilde{x}^{(f)}$ and interval is the allowable interval of the values of $\tilde{\boldsymbol{x}}$.

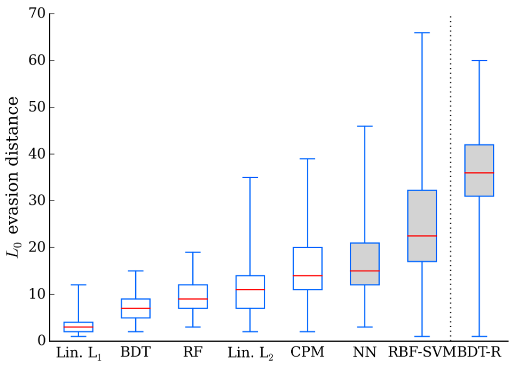
The complexity of the algorithm 3 is $O(|t.nodes| \log |t.nodes|)$. Once the list of tuple u has been calculated for each tree, the leaf prediction of $\boldsymbol{x}$ is subtracted to have the score variation between $\tilde{\boldsymbol{x}}$ and $\boldsymbol{x}$. Next, the tuple *dimension-interval-variation* that generates the largest variation $\mathcal{T}(\tilde{\boldsymbol{x}}) - \mathcal{T}(\boldsymbol{x})$ is calculated. The final search costs $O(|U| \log |U|)$, with $U = \{u_1, \ldots, u_{|\mathcal{T}|}\}$. By construction $|U|$ is at most $\sum_{t\in\mathcal{T}} |t.leaves|$. The computational complexity of the whole approximated method is $O(|\mathcal{T}| \log |\mathcal{T}|)$, much less than that of the optimal evasion.

3.2.3 BDT-R model

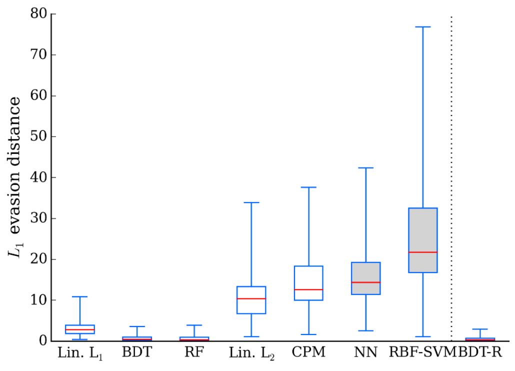
In [32] they showed empirically how training a BDT, without loss of predictive accuracy, increasing the training set with evasion instances produced with symbolic prediction 3.2.2 at each boosting round. This training technique has been called *adversarial boosting* and the generated model is called Boosted Decision Tree - Robust. In particular given an instance $\boldsymbol{x}$ with label y and a modification budget $b \geq 1$, a budgeted adversarial training instance $\boldsymbol{x}'$, it is such that $\| \boldsymbol{x} - \boldsymbol{x}' \|_0 \leq b$, and the margin $y\mathcal{T}(\boldsymbol{x}')$ is as small as possible. Since the model uses the BDT ensemble learning algorithm, at each iteration the created base-learner tries to correct the ensemble errors. For each iteration, the base-learner is trained with the normal training dataset combined with the evading instances created by attacking the ensemble. Consequently, the base-learner corrects the errors made by the ensemble on both normal and evading instances. It is important to report, as mentioned in [32], the BDT-R model trained with L_0 evading instances, has less robustness against attacks based on the distances L_1, L_2 and L_∞, than that of a normal trained BDT. Figure 3.1 shows the results of the experiments conducted in [32] to evaluate the performance of BDT-R compered to other models. The performances of the various models are measured with respect to attacks generated with the distance functions: L_0 in 3.1a, L_1 in 3.1b, L_2 in 3.1c and L_∞ in 3.1d. The boxes represent evasion bounds for different metrics, the white boxes represent the optimal attacks, while the gray boxes the best-effort attacks (how much effort an attacker must make to break the model). The smallest bounds, 25-50% and 50-75% quartiles and largest bounds are shown. The red line represents the average of the score. The higher the average, the more perturbations are needed to evade the model. As can be seen from the results and as explained above, the BDT-R model trained to be robust to attacks generated with distance function L0 has very low performances when it classifies attacks generated with different norms.

3.3 Random Subspace Method

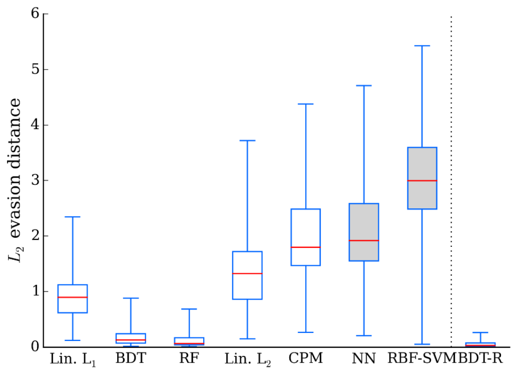
The latest learning algorithm proposed in this chapter is the Random Subspace Method (RSM) [29], an ensemble methods to train a robust model against evasion attack [6]. RSM is very similar to Bagging, but unlike the latter, it performs the boostrap sampling on the feature space and not on the instances. Thanks to the use of a subset of features for each base-learner, RSM performs well in contexts of datasets with a high dimensionality. The large number of possible subspaces of features allow to overcome the problem of the curse of dimensionality present in many models [29]. Originally RSM was born to improve the base-learner's performance from a point of view of accuracy in operation phase, not considering an attack context. In fact, there are several articles that show how RSM improves the performance of base-learners. For example, in [43] it is used for linear models, in [46] for support vector machine [17], while in the articles [28, 29] it is used with decision trees. In particular, in [28] it is empirically shown that RSM works well on binary trees. In the context of the decision trees, it was shown how randomly



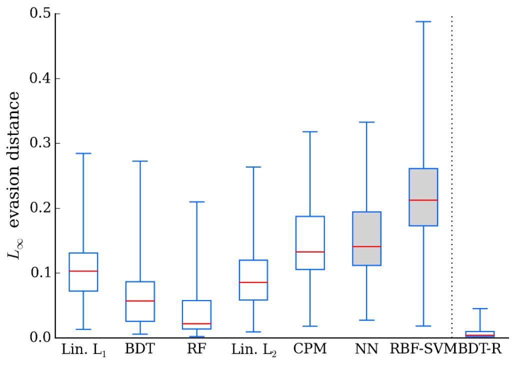
(a) Evasion attacks with L_0-norm.



(b) Evasion attacks with L_1-norm.



(c) Evasion attacks with L_2-norm.



(d) Evasion attacks with L_∞-norm.

Figure 3.1: Comparison between BDRT-R and other learning algorithms against evasion attacks generated with: L_0-norm, L_1-norm, L_2-norm and L_∞-norm. The images of the results were taken from [32].

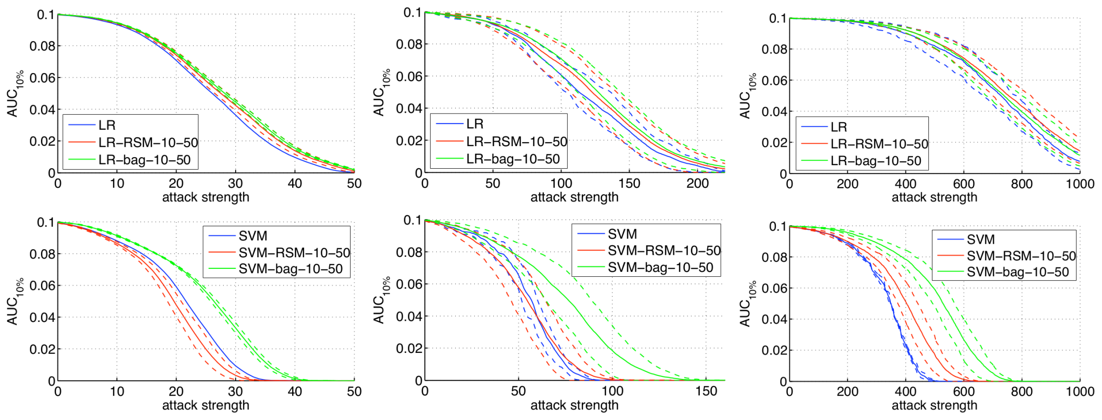
choosing a subset of features leads to an improvement in generalization accuracy, while maintaining the performance on training data unchanged. With RSM, each tree within the ensemble generalizes differently from the others.
In literature there are references to the fact that improving the robustness of a model through the use of an ensemble does not always lead to good results. The reason why in [5, 6] they investigate the robustness produced by RSM against adversarial examples is the following. In [33] they assume that to increase the robustness of a linear model it is necessary to distribute the features according to weight (the discriminative importance) in the most uniform way. So doing the adversary is forced to attack a greater number of features to evade the model. If the attacker knows what the most important features are within the training

set, he can try to perform the attack only on those. If, as reported in [33], the learning algorithm does not over-emphasize (under-emphasise) features which are highly (slightly) discriminant on training samples, the attacker has no way to use this strategy. On the other hand, the distribution of the weight of the features in a uniform way can lead to a decrease in the model's performance on normal instances. One property of the RSM is that thanks to the randomness of the boostrap sampling, very discriminative features end up in different base-learners compared to the less discriminative ones. This separation allows less representative features not to be excluded (or considered less) in the training phase due to more important features. This leads to the creation of new ways to discriminate training instances. From what has been said above, RSM is suitable for reaching the trade-off between acuracy in operation phase and for producing the robustness strategy proposed in [33] mentioned above. In [5] they confirmed empirically how RSM distributes the weights of the features in a more uniform way than the other models compared and this has led to an improvement in the robustness under attack. This effect is highly dependent on the size of the feature set and the size of the ensemble. The pseudocode of the RSM algorithm is reported in 4. Finally below, we reported

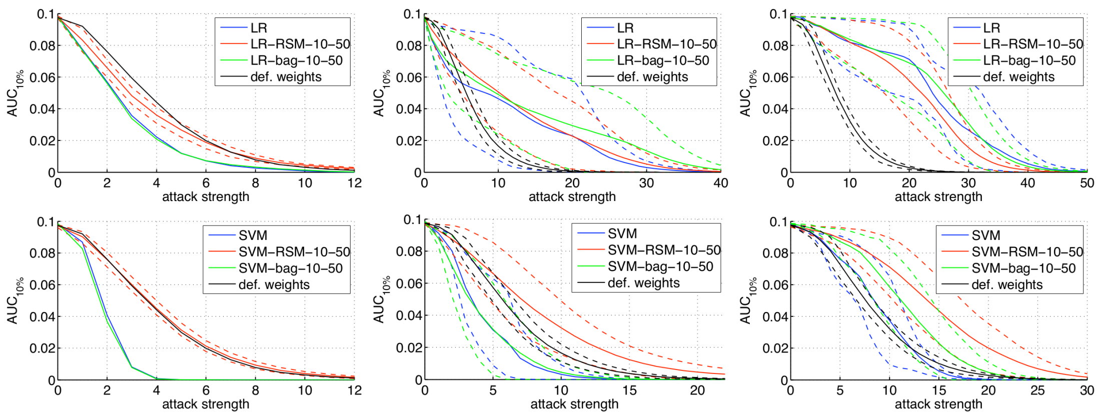
Algorithm 4 RandomSubspaceMethod

```
Require: A training set $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_n, y_n)\}$, the features space $\mathcal{F}$, the
    learning algorithm $\mathfrak{L}$, the ensemble size $r$ and the percentage of features
    $p \in [0, 1]$.
1: function RandomSubspaceMethod($\mathcal{D}, \mathcal{F}, \mathfrak{L}, r, p$)
2:     $\mathcal{T} \leftarrow \emptyset$
3:     for $i = 1$ to $r$ do
4:         $\mathcal{F}_i \subset \mathcal{F} \wedge |\mathcal{F}_i| = \lfloor |\mathcal{F}| p \rfloor$
5:         $t_i \leftarrow \mathfrak{L}(\mathcal{D}, \mathcal{F}_i)$
6:         $\mathcal{T} \leftarrow \mathcal{T} \cup t_i$
7:     end for
8:     return $\mathcal{T}$
9: end function
```

some of the results of the analyses conducted in [5]. Figures in 3.2 compare the robustness of the RSM ensemble learning algorithm with other strategies, as the attacker's strength increases. The experiments performed concern a spam filtering task on the TREC 2007 email corpus [5]. Plots in the figure 3.2a represent the performance of text classifiers in which features correspond to words in emails. while plots in figure 3.2b shows the performance of the SpamAssassin filter whose decision function weights have been set by each of the learning algorithm analyzed. For each figure, the weak-learners used are LR (top) and SVM (bottom). while the ensemble methods used are RSM and Bagging. The performances of the ensemble methods were also compared with the single weak-learner to see if the ensembling improves the strength of the weak-leaner. The three plots for each weak-learner represent, left: worst-case attack; middle: non-worst-case attack; right: random attack. The evaluation metric is $\mathrm{AUC}_{10\%}$ and for each model it is represented by a solid line, while the standard deviation for each curve is represented through two dashed lines ($\pm$ std).



(a) Comparison between text classifiers in which features correspond to words in emails.



(b) Comparison of SpamAssassin filter results with the decision function weights set with the analyzed learning algorithms.

Figure 3.2: Plots of RSM performances compared to other learning algorithms. The images were taken from [5].

3.4 Robust Split

This section report another training algorithm for robust decision trees taken from the state of the art of aversarial machine learning. The algorithm is called *Robust Split* [13], which obtains reliable models robust against adversarial attacks, through the formulation of the decision tree training process as an optimization problem over finding best splitting point. Robust Split take the distance between data points into accounts and try to optimize the worst case performance under adversarial perturbations. In Figure 3.3 we have given an example taken from [13]. The example is very simple but perfectly explains the concept behind the robust split algorithm.

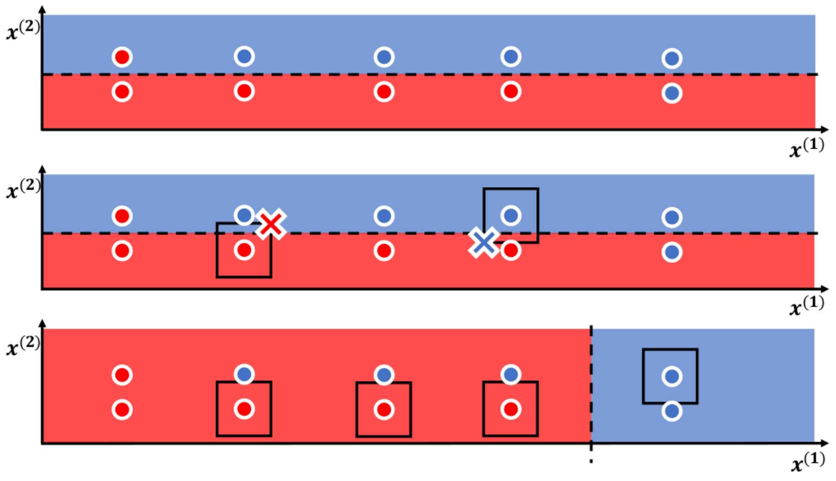


Figure 3.3: Given 10 points as in figure, in a two-dimensional space a model trained with a classic DT algorithm obtains inferior performance under attack, compared to the Robust Split model. Upper: The classical learning algorithm separates the 10 points simply with a horizontal line on the feature $x^{(2)}$. The accuracy produced by this split without attack is 0.8. Middle: The split that generates the best accuracy on safe data is not able to separate the ℓ_∞ balls, norm bounded noise (square boxes), around the instances. Therefore an attacker can push instances beyond the decision boundary through a perturbation within ℓ_∞. The minimum accuracy under attack is 0. Lower: The Robust Split finds a differente splitting point, on the feature $x^{(1)}$. The attacker can no longer push an instance beyond the decision boundary. Robust Split gets 0.7 on both safe and attacked input [13].

3.4.1 Attacker Model

The attacker model performs adversarial example attacks whose perturbations are not perceptible to the human eye, in a white box context. The model is defined on a budget ϵ which limits the strength of the attacker and specifies how much each feature of an instance can be perturbed. More formally, let A be the attacker, the attack is defined on the ℓ_∞ ball of radius ϵ around an instance $\boldsymbol{x} \in \mathcal{X}$. So, given an instance $\boldsymbol{x}$, the attacker A can generate any perturbed instance $\boldsymbol{x'} \in A_\epsilon^\infty(\boldsymbol{x})$, where $A_\epsilon^\infty(\boldsymbol{x}) = [x^{(1)} - \epsilon, x^{(1)} + \epsilon] \times \cdots \times [x^{(d)} - \epsilon, x^{(d)} + \epsilon]$.

3.4.2 Model Definition

To understand the Robust Split algorithm it is fundamental to view the Table 3.3, containing the definitions given in [13], of the sets used during the explanation. Let

Table 3.3: Definition of the notions of the robust learning algorithm proposed in [13]. The variables f and v are respectively the feature and the threshold on which the split is being performed.

Notation	Definition
$\mathcal{I}$	set of instances on the current node, $\mathcal{I} \subseteq \mathcal{D}$
$\mathcal{I}_0$	$\mathcal{I} \cap \{(\boldsymbol{x}_i, y_i) \mid y_i = 0\}$ (for classification)
$\mathcal{I}_1$	$\mathcal{I} \cap \{(\boldsymbol{x}_i, y_i) \mid y_i = 1\}$ (for classification)
$\mathcal{I}_L$	$\mathcal{I} \cap \{(\boldsymbol{x}_i, y_i) \mid x_i^{(f)} < v\}$
$\mathcal{I}_R$	$\mathcal{I} \cap \{(\boldsymbol{x}_i, y_i) \mid x_i^{(f)} \geq v\}$
$\Delta\mathcal{I}$	$\mathcal{I} \cap \{(\boldsymbol{x}_i, y_i) \mid v - \epsilon \leq x_i^{(f)} \leq v + \epsilon\}$
$\Delta\mathcal{I}_L$	$\Delta\mathcal{I} \cap \mathcal{I}_L$
$\Delta\mathcal{I}_R$	$\Delta\mathcal{I} \cap \mathcal{I}_R$
$\mathcal{I}_L^o$	$\mathcal{I}_L \setminus \Delta\mathcal{I}$
$\mathcal{I}_R^o$	$\mathcal{I}_R \setminus \Delta\mathcal{I}$

$\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=0}^N$ be the dataset, with $y_i \in \mathbb{R}$ (or $y_i \in \{0,1\}$ for binary classification) and $\boldsymbol{x}_i \in \mathbb{R}^d$, with the feature values normalized into $[0,1]$. Let $f \in [1,d]$ and $v \in \mathbb{R}$ be the splitting point of the node. The Robust Split algorithm wants to maximize the minimum performance of the tree under attack. Through the following max-min optimization problem 3.10 it is possible to find the best parameters f^* and v^* that divide the dataset and maximize the lowest performances under attack:

$$\begin{aligned} f^*, v^* &= \arg\max_{f,v} \underbrace{\min_{\mathcal{I}'} \texttt{score}(f, v, \mathcal{I}')}_{\texttt{robust\_score}(f,v,\mathcal{I})} \\ &\text{s.t. } \mathcal{I}' = \{(\boldsymbol{x}', y) \mid \forall (\boldsymbol{x}, y) \in \mathcal{I} \wedge \boldsymbol{x}' \in A_\epsilon^\infty(\boldsymbol{x})\} \\ &\text{with respect to } f \text{ and } v \end{aligned} \tag{3.10}$$

For every possible pair of (f, v), the worst case perturbation set $\mathcal{I}'$ is calculated. `Robust_score` represents the worst case of the `score` function obtained by the model under attack, with splitting point (f, v). The optimization problem 3.10 calculates $\mathcal{I}'$ for all instances of $\mathcal{I}$, even on those that are too far away to pass the decision boundary. In fact, given f and v, only instances with $x_i^{(f)} \in [v-\epsilon, v+\epsilon]$ can be efficiently attacked. Thus $\mathcal{I}$ can be divided into three subsets: $\mathcal{I}_L^o$ and $\mathcal{I}_R^o$ are the sets of instances that go left if $x_i^{(f)} < v$ and right if $x_i^{(f)} \geq v$ respectively, regardless of the attacker A. While $\Delta\mathcal{I}$ is the *ambiguity set*, and contains the instances that can cross the threshold v if perturbed by A. Thanks to this division and the introduction of the variables $s_i \in \{0, 1\}$ they have transformed `robust_score` of the problem 3.10 into a 0-1 integer optimization problem with $|\Delta\mathcal{I}|$ variables:

$$\begin{aligned} f^*, v^* = \arg\max_{f,v} & \underbrace{\min_{s_i \in \{0,1\}} \texttt{score}(\mathcal{I}'_L, \mathcal{I}'_R)}_{\texttt{robust\_score}(f,v,\mathcal{I})} \\ \text{s.t. } \mathcal{I}'_L &= \mathcal{I}_L^o \cup \{(\boldsymbol{x}_i, y_i) \in \Delta\mathcal{I} | s_i = 0\}, \\ \text{and } \mathcal{I}'_R &= \mathcal{I}_R^o \cup \{(\boldsymbol{x}_i, y_i) \in \Delta\mathcal{I} | s_i = 1\}. \end{aligned} \tag{3.11}$$

Even if the search for the best split is done only on instances in $\Delta\mathcal{I}$ a 0-1 integer optimization problem is NP-Hard in general. In [13] two approximations are given to this problem, one based on information gain 1.2 and one on GBDT 1.3.2.

Robust Splitting with Information Gain Score

The first approximation proposed is based on a decision tree for binary classification $y \in \{0, 1\}$, whose node split is based on the informaiton gain. The `score` function in 3.10 is defined as follows:

$$\texttt{score}(f, v, \mathcal{I}) = IG(f, v) = H(y) - H(y|x^{(f)} < v), \tag{3.12}$$

where

$$H(y) = -\frac{|\mathcal{I}_0|}{|\mathcal{I}|} \log\left(\frac{|\mathcal{I}_0|}{|\mathcal{I}|}\right) - \frac{|\mathcal{I}_1|}{|\mathcal{I}|} \log\left(\frac{|\mathcal{I}_1|}{|\mathcal{I}|}\right), \tag{3.13}$$

and

$$\begin{aligned} & H(y|x^{(f)} < v) = \\ & -\frac{|\mathcal{I}_L|}{|\mathcal{I}|}\left[\frac{|\mathcal{I}_L \cap \mathcal{I}_0|}{|\mathcal{I}_L|} \log\left(\frac{|\mathcal{I}_L \cap \mathcal{I}_0|}{|\mathcal{I}_L|}\right) - \frac{|\mathcal{I}_L \cap \mathcal{I}_1|}{|\mathcal{I}_L|} \log\left(\frac{|\mathcal{I}_L \cap \mathcal{I}_1|}{|\mathcal{I}_L|}\right)\right] \\ & -\frac{|\mathcal{I}_R|}{|\mathcal{I}|}\left[\frac{|\mathcal{I}_R \cap \mathcal{I}_0|}{|\mathcal{I}_R|} \log\left(\frac{|\mathcal{I}_R \cap \mathcal{I}_0|}{|\mathcal{I}_R|}\right) - \frac{|\mathcal{I}_R \cap \mathcal{I}_1|}{|\mathcal{I}_R|} \log\left(\frac{|\mathcal{I}_R \cap \mathcal{I}_1|}{|\mathcal{I}_R|}\right)\right]. \end{aligned} \tag{3.14}$$

In [13] they showed that if $\frac{|\mathcal{I}_L \cap \mathcal{I}_0|}{|\mathcal{I}_0|} < \frac{|\mathcal{I}_L \cap \mathcal{I}_1|}{|\mathcal{I}_1|}$ and $\frac{|\mathcal{I}_L \cap \mathcal{I}_0|+1}{|\mathcal{I}_0|} \leq \frac{|\mathcal{I}_L \cap \mathcal{I}_1|}{|\mathcal{I}_1|}$, perturb an instance from $\Delta\mathcal{I}_R$ with label 0 to $\Delta\mathcal{I}_L$, information gain decreases. Through this theorem it is possible to find the adversary's perturbation direction to minimize the information gain. To minimize information gain, the attacker must perturb the instances in $\Delta\mathcal{I}$, so that $\frac{|\mathcal{I}_L \cap \mathcal{I}_0|}{|\mathcal{I}_0|} \approx \frac{|\mathcal{I}_L \cap \mathcal{I}_1|}{|\mathcal{I}_1|}$. The solution they have proposed minimizes $\left|\frac{|\mathcal{I}_L \cap \mathcal{I}_0|}{|\mathcal{I}_0|} - \frac{|\mathcal{I}_L \cap \mathcal{I}_1|}{|\mathcal{I}_1|}\right|$ as an approximation and upper bound to the optimal solution, with a time complexity of $O(dN^2)$.

Robust Splitting with GBDT models

Subsequently they employed their robust splitting strategy in the tree boosting setting. In this case multiple robust decision trees are created to increase the strength of the ensemble. In particular, they used their robust splitting technique in the split procedure of the nodes of the GBDT ensemble learning algorithm. The changes only influenced the calculation of the score for the choice of the best split. The `score` function in 3.10 is defined as:

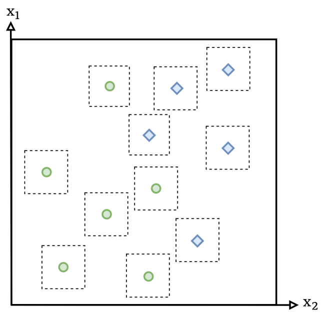
$$\texttt{score}(f, v, \mathcal{I}) = S(\mathcal{I}_R, \mathcal{I}_L) \tag{3.15}$$

where $S(\mathcal{I}_R, \mathcal{I}_L)$ is the score function used for training a normal GBDT. Also in this case to minimize $S(\mathcal{I}_R, \mathcal{I}_L)$ it is necessary to try all the permutations of the instances in $\Delta\mathcal{I}$ between the two sets $\mathcal{I}_R$ amd $\mathcal{I}_L$. Finding the permutation that minimizes the score is like solving the 0-1 integer optimization problem 3.11, for every possible pair of (f, v) which is actually NP-Hard. Therefore they decided to approximate this problem through four representative cases:

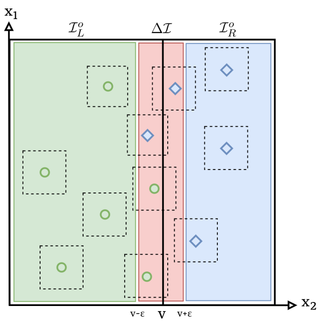
$$\begin{aligned}\texttt{robust\_score}(f, v, \mathcal{I}) \approx \min\{&S(\mathcal{I}_L, \mathcal{I}_R), S(\mathcal{I}_L^o \cup \Delta\mathcal{I}, \mathcal{I}_R^o),\\ &S(\mathcal{I}_L^o, \mathcal{I}_R^o \cup \Delta\mathcal{I}), S(\mathcal{I}_L^o \cup \Delta\mathcal{I}_R, \mathcal{I}_R^o \cup \Delta\mathcal{I}_L)\}.\end{aligned} \tag{3.16}$$

For each f and v, the four scores are defined as: The first, $S(\mathcal{I}_L, \mathcal{I}_R)$ represents the score of non-perturbed instances. The second, $S(\mathcal{I}_L^o \cup \Delta\mathcal{I}, \mathcal{I}_R^o)$ is the score when the attacker moves all the instances of $\Delta\mathcal{I}$ to the left. The third, $S(\mathcal{I}_L^o, \mathcal{I}_R^o \cup \Delta\mathcal{I})$ is the score when the attacker moves all the instances of $\Delta\mathcal{I}$ to the right. The fourth: $S(\mathcal{I}_L^o \cup \Delta\mathcal{I}_R, \mathcal{I}_R^o \cup \Delta\mathcal{I}_L)\}$ is the score when the attacker moves all instances of $\Delta\mathcal{I}_R$ to the left and all the instances of $\Delta\mathcal{I}_L$ to the right. The approximation takes place by taking the minimum of the four scores as a solution.
Below we given a figurative explanation of the effects of the four partitions to approximate the worst attack in the case of GBDT.

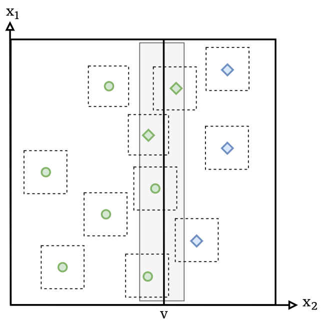
The figures in 3.4 show the possible effects of the four partitions used in the approximation. The points inside the figure represent two types of objects, circles from the green class and diamonds from the blue class. Figure 3.4a represents the points inside $\mathcal{I} \subseteq \mathcal{D}$ as input to the current node. In Figure 3.4b the model chooses to test the data with respect to $f = 2$ and then to divide the data with a



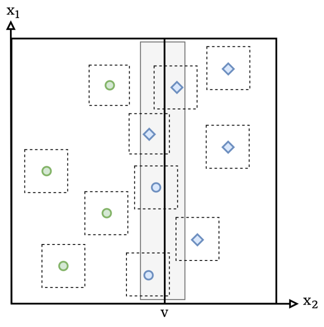
(a) $\mathcal{I}$



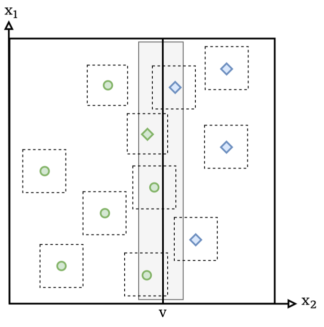
(b) $\mathcal{I}_L$ and $\mathcal{I}_R$



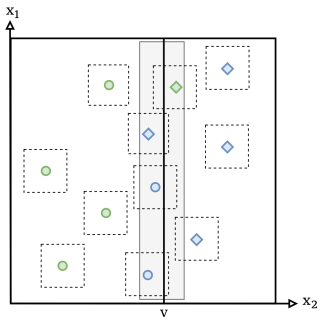
(c) $\mathcal{I}_L^o \cup \Delta\mathcal{I}, \mathcal{I}_R^o$



(d) $\mathcal{I}_L^o, \mathcal{I}_R^o \cup \Delta\mathcal{I}$



(e) $\mathcal{I}_L, \mathcal{I}_R$



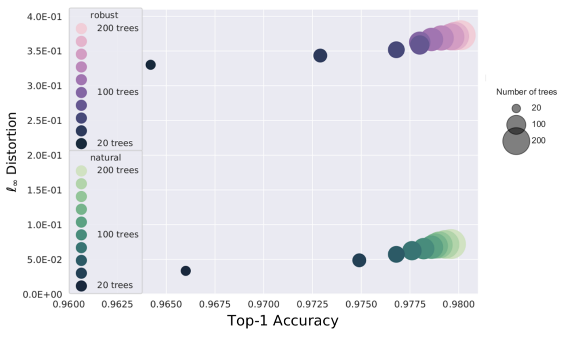
(f) $\mathcal{I}_L^o \cup \Delta\mathcal{I}_R, \mathcal{I}_R^o \cup \Delta\mathcal{I}_L$

Figure 3.4: `robust_score` for boosted tree - approximation of the worst attack.

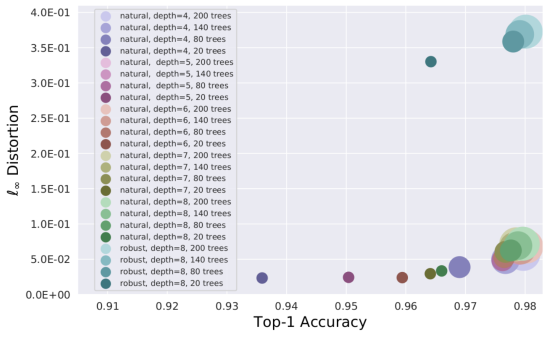
threshold v with respect to the values of $x^{(2)}$. The data are divided into sets: $\mathcal{I}_L^o$ which contains all the instances to the left of the split and which cannot cross it, $\mathcal{I}_R^o$ contains the objects to the right of the split and which cannot cross it and finally $\Delta\mathcal{I}$ which contains instances that can cross the split because they have the value $v - \epsilon \leq x^{(2)} \leq v + \epsilon$. The partition $\mathcal{I}_L^o \cup \Delta\mathcal{I}, \mathcal{I}_R^o$ in Figure 3.4c produces a division with two errors, two diamonds are classified as greens. The partition $\mathcal{I}_L^o, \mathcal{I}_R^o \cup \Delta\mathcal{I}$ in Figure 3.4d also produces two errors since two circles are classified as blue. The partition $\mathcal{I}_L, \mathcal{I}_R$ in Figure 3.4e leads to an error because one diamond is classified as green. Finally the partition $\mathcal{I}_L^o \cup \Delta\mathcal{I}_R, \mathcal{I}_R^o \cup \Delta\mathcal{I}_L$ in Figure 3.4f causes tree errors because two circles are classified as green and one diamond is classified as blue. The model is trained by choosing the partition that generates the highest number of mistakes and therefore generates the lowest score.
In algorithm 5, the best split pseudocode proposed in [13] for the GDBT-based approximation is given. The pseudocode only shows how to best split instances in $\mathcal{I}$. In a complete implementation, the rest of the algorithm is the same used in a normal GBDT. Finally in Figures 3.5 and 3.6 we reported some results of the experiments performed in [13], which show how to train a GBDT with `robust_score` increases the robustness to L_∞-norm evasion attack compared to a normal GBDT.

Algorithm 5 ROBUST_SPLIT_FOR_BOOSTED_TREE

```
Require: The instances set $\mathcal{I}$ of the current node and the radius $\epsilon$ of the $\ell_\infty$ ball.
 1: function Robust_Split_for_Boosted_Tree($\mathcal{I}, \epsilon$)
 2:     for $f \leftarrow 1$ to $d$ do
 3:         for $m$ in sorted($\mathcal{I}$, ascending order by $x_m^{(f)}$) do
 4:             $v \leftarrow \frac{1}{2}(x_m^{(f)} + x_{m+1}^{(f)})$
 5:             $\mathcal{I}_L^o \leftarrow \{(\boldsymbol{x}_i, y_i) | x^{(f)} < v - \epsilon\}, \Delta\mathcal{I}_L \leftarrow \mathcal{I} \cap \{(\boldsymbol{x}_i, y_i) | v - \epsilon \leq x^{(f)} < v\}$
 6:             $\mathcal{I}_R^o \leftarrow \{(\boldsymbol{x}_i, y_i) | x^{(f)} > v + \epsilon\}, \Delta\mathcal{I}_R \leftarrow \mathcal{I} \cap \{(\boldsymbol{x}_i, y_i) | v \leq x^{(f)} < v + \epsilon\}$
 7:             $S_1 = S(\mathcal{I}_L, \mathcal{I}_R)$
 8:             $S_2 = S(\mathcal{I}_L^o \cup \Delta\mathcal{I}, \mathcal{I}_R^o)$
 9:             $S_3 = S(\mathcal{I}_L^o, \mathcal{I}_R^o \cup \Delta\mathcal{I})$
10:             $S_4 = S(\mathcal{I}_L^o \cup \Delta\mathcal{I}_R, \mathcal{I}_R^o \cup \Delta\mathcal{I}_L)$
11:             robust_score($f, v$) $\leftarrow \min\{S_1, S_2, S_3, S_4\}$
12:         end for
13:     end for
14:     $f^*, v^* \leftarrow \arg\max_{f,v}$ robust_score($f, v$)
15:     return split on feature $f^*$ with a threshold $v^*$
16: end function
```



(a) robustness vs. classification accuracy



(b) distortion vs. classification accuracy

Figure 3.5: In the figure we have reported some of the results of the experiments conducted [13] on dataset MNIST to compare the performance of natural GBDT and robust GBDT (with robust training parameter $\epsilon = 0.3$) against adversarial examples found by Cheng's L_∞ attack [16]. Figure (a) show robustness vs. classification accuracy plot of GBDT models with different depth and different numbers of trees. Figure (b) shows distortion vs. classification accuracy plot of GBDT models with different numbers of trees. In both plots it can be seen that the robust version of GBDT maintains a higher accuracy under attack compared to the normal version. The images of the experiments were taken from [13].

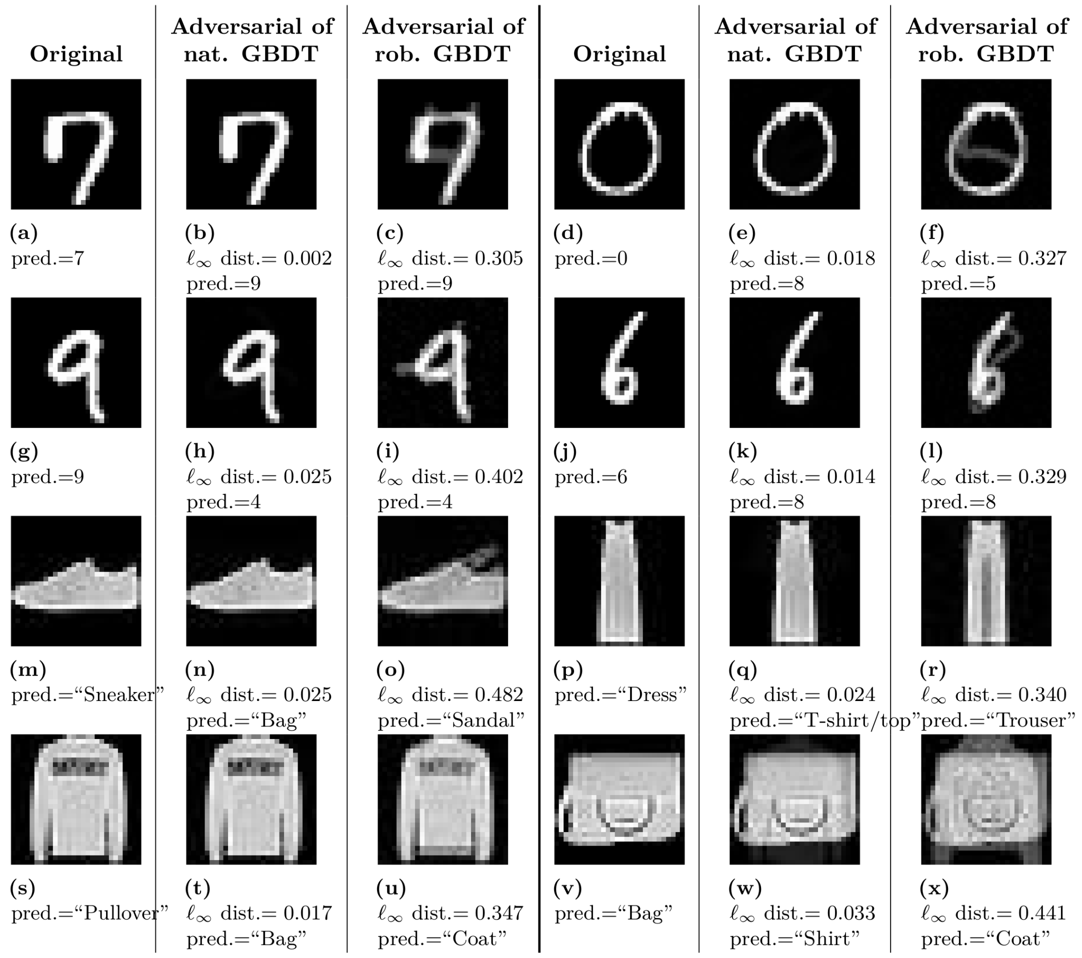

Original
Adversarial of nat. GBDT
Adversarial of rob. GBDT
Original
Adversarial of nat. GBDT
Adversarial of rob. GBDT
(a) pred.=7
(b) ℓ_∞ dist.= 0.002 pred.=9
(c) ℓ_∞ dist.= 0.305 pred.=9
(d) pred.=0
(e) ℓ_∞ dist.= 0.018 pred.=8
(f) ℓ_∞ dist.= 0.327 pred.=5
(g) pred.=9
(h) ℓ_∞ dist.= 0.025 pred.=4
(i) ℓ_∞ dist.= 0.402 pred.=4
(j) pred.=6
(k) ℓ_∞ dist.= 0.014 pred.=8
(l) ℓ_∞ dist.= 0.329 pred.=8
(m) pred.="Sneaker"
(n) ℓ_∞ dist.= 0.025 pred.="Bag"
(o) ℓ_∞ dist.= 0.482 pred.="Sandal"
(p) pred.="Dress"
(q) ℓ_∞ dist.= 0.024 pred.="T-shirt/top"
(r) ℓ_∞ dist.= 0.340 pred.="Trouser"
(s) pred.="Pullover"
(t) ℓ_∞ dist.= 0.017 pred.="Bag"
(u) ℓ_∞ dist.= 0.347 pred.="Coat"
(v) pred.="Bag"
(w) ℓ_∞ dist.= 0.033 pred.="Shirt"
(x) ℓ_∞ dist.= 0.441 pred.="Coat"

Figure 3.6: In figure we reported some of the results of the experiments conducted in [13] on MNIST and Fashion-MNIST dataset. The experiments compared the performance of a natural GBDT and robust GBDT (with robust training parameter $\epsilon = 0.3$) against adversarial examples found by Cheng's L_∞ attack [16]. Both models consist of 200-tree gradient boosted decision tree. The experiment shows that robust GBDT requires a very strong attack to be fooled, such as to be evident to the human eye. Instead for normal GBDT imperceptible perturbations are enough to completely fool the model. The images of the experiments were taken from [13].

3.5 Training Evasion-Aware Decision Trees

In [12] a new method has been proposed for learning a decision tree that is at the same time accurate and nearly insensitive to evasion attacks. The learnering algorithm is called the *Treant* (Training Evasion-Aware Decision Trees) and is used as a base-learning algorithm to train the base-learners of a random forest. Before going into the definition of the learning algorithm it is important to define the attacker model for which the model wants to be robust.

3.5.1 Attacker Model

Let A be the attacker, he has complete knowledge of the model (white box attacks) and generates evasion attacks that are not perceptible to the human eye. From the definition given in [12], the model of the attacker A is defined with a pair of (R, K) where R is the set of *rewriting rules* and K is the available budget. The rules in R define how the instances can be modified and $K \in \mathbb{R}^+$ is the amount of the budget that the attacker can spend on performing the attack. The budget limits the corruptions that the attacker can perform on an instance. Each rule $r \in R$ has the following form:

$$[a, b] \xrightarrow{f}_k [\delta_l, \delta_u], \tag{3.17}$$

with $[a, b]$ and $[\delta_l, \delta_u] \in \mathbb{R} \cup \{-\infty, +\infty\}$. The interval $[a, b]$ indicates the condition for applying the rule. If the value $x^{(f)} \in [a, b]$ and the attackers has enough budget to pay the cost k of the rule, then he can alter the instance $\boldsymbol{x}$ by adding to $x^{(f)}$ a value $v \in [\delta_l, \delta_u]$. The attacker can use all the rewriting rules he wants, as long as he has available budgets. Through the definition of the attacker model, let $A(\boldsymbol{x})$ be the set of all possible perturbations $\boldsymbol{x}' \in A(\boldsymbol{x})$ of $\boldsymbol{x}$ that the attacker can perform with the rewriting rules R and budget K. Let $\mathcal{D}' = A(\mathcal{D})$ be the set of all instances of $\mathcal{D}$ perturbed by the attacker. In other words, for each pair $(\boldsymbol{x}, y) \in \mathcal{D}$ all pairs $(\boldsymbol{x}', y)$, with $\boldsymbol{x}' \in A(\boldsymbol{x})$, are in $\mathcal{D}'$. This rule-based attacker definition, allows a simple generalization to categorical variables, to model also asymmetric perturbations and to cover or approximate the standard distanced-based models.

3.5.2 Model Defintion

In general the Treant algorithm does not differ much from the learning algorithm of a normal DT. What guarantees the robustness of the model is the way in which the dataset is divided during the growth phase of the tree When the dataset is divided, the algorithm ensures that the loss generated by the division is the lowest compared to the maximum loss generated by all possible attacks. In particular the model is defined on the following optimization problem [37]:

$$\hat{h} = \underset{h \in \mathcal{H}}{\arg\min} \underbrace{\max_{\mathcal{D}' \in A(\mathcal{D})} \mathcal{L}(h, \mathcal{D}')}_{\mathcal{L}^A(h, \mathcal{D})} . \tag{3.18}$$

The loss $\mathcal{L}^A(h, \mathcal{D})$ is called *loss under attack*, which represents the maximum loss generated by A through corruption of $\mathcal{D}$ with respect to the model h. The purpose of the training algorithm is to find the model h which minimizes the loss under attack. In other words, minimizing the strongest attack the adversary can make against any possible h. The double optimization problem finds the model h, whose maximum loss generated by the attacker, is the lowest. The external part of the problem refers to the principle of empirical risk minimizaton 1.1, which tries to find the hypothesis that minimizes the $\mathcal{L}^A$ of the training set. They proposed a way to improve tree construction compared to the traditional greedy method, based on node splitting. The method involves minimizing the $\mathcal{L}^A$ during each growth step of the tree, in this way the tree grows as long as the maximum error that the model commits under attack decreases. The algorithm proposed in [12], guarantees the robustness of the model using two mechanisms, the *robust splitting* and the *attack invariance.*

Robust splitting

The robust splitting strategy optimizes the $\mathcal{L}^A$ at each growth step of the tree. The creation of the robust tree is made recursively, and the search for the hypothesis $\hat{h}$ of 3.18 corresponds to the node $\sigma(f, v, \lambda(\hat{y}_l), \lambda(\hat{y}_r))$ which generates the lower $\mathcal{L}^A$. So for every possible pair of (f, v) the problem of 3.18 must find the value of the leaves $\hat{y}_l$ and $\hat{y}_r$ that minimizes the $\mathcal{L}^A$. The search for the best $\sigma(f, v, \lambda(\hat{y}_l), \lambda(\hat{y}_r))$ is done through the *ternary partitioning.* The ternary partitioning divide the dataset $\mathcal{D}$ into three subset. The subset $\mathcal{D}_l(f, v, A)$ represents the instances of $\mathcal{D}$ which always fall in the left branch, regardless of the attacks produced by A. The correspondent for the right branch is $\mathcal{D}_r(f, v, A)$. The third subset $\mathcal{D}_u(f, v, A)$, where $\boldsymbol{x} \in \mathcal{D}_u(f, v, A) \iff \exists \boldsymbol{x}', \boldsymbol{x}'' \in A(\boldsymbol{x}) | x'^{(f)} \leq v \wedge x''^{(f)} l > v$. Thus, D_u contains instances that can change direction due to the attack. The search for the $\mathcal{L}^A$ should be done only on $\mathcal{D}_u(f, v, A)$, since they are the only instances for which the attacker has the intention to modify. For $\mathcal{D}_l(f, v, A)$ and $\mathcal{D}_r(f, v, A)$ the loss $\mathcal{L}^A = \mathcal{L}$. Problem 3.18 can be rewritten as:

$$(\hat{y}_l, \hat{y}_r) = \underset{y_l, y_r}{\arg\min}\, \mathcal{L}^A(\sigma(f, v, \lambda(y_l), \lambda(y_r)), \mathcal{D}), \tag{3.19}$$

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

$$\begin{aligned} &\mathcal{L}^A(\sigma(f, v, \lambda(y_l), \lambda(y_r)), \mathcal{D}) = \\ &= \mathcal{L}(\lambda(y_l), \mathcal{D}_l(f, v, A)) + \\ &+ \mathcal{L}(\lambda(y_r), \mathcal{D}_r(f, v, A)) + \\ &+ \sum_{(\boldsymbol{x}, y) \in \mathcal{D}_l(f, v, A)} \max\{\ell(y_l, y), \ell(y_r, y)\}. \end{aligned} \tag{3.20}$$

Inside the sum in 3.20, for the calculation of the loss under attack, the highest loss is taken between $\ell(y_l, y)$ and $\ell(y_r, y)$. This is because the attacker always chooses to send the instance in the direction that causes the greatest loss. It is important to note that if the instance-level loss ℓ is convex, then also $\mathcal{L}^A$ will be convex and therefore can be efficiently optimized. Let $\mathcal{N}$ be the set of nodes generated for each pair (f, v), defined as follow:

$$\begin{aligned} \mathcal{N} \leftarrow \{\sigma(f, v, \lambda(\hat{y}_l), \lambda(\hat{y}_r)) | f \in [1, d] \wedge \exists (\boldsymbol{x}, y) \in \mathcal{D} : x^{(f)} = v \wedge \\ \wedge \ \hat{y}_l, \hat{y}_r = \underset{y_l, y_r}{\arg\min}\, \mathcal{L}^A(\sigma(f, v, \lambda(y_l), \lambda(y_r)), \mathcal{D})\}. \end{aligned} \tag{3.21}$$

Given 3.19, to calculate the node $\hat{t} = \sigma(f, v, \lambda(\hat{y}_l), \lambda(\hat{y}_r))$ which produces the lowest $\mathcal{L}^A$, the following optimization problem must be solved:

$$\hat{t} = \underset{t \in \mathcal{N}}{\arg\min}\, \mathcal{L}^A(t, \mathcal{D}) = \sigma(f, v, \lambda(\hat{y}_l), \lambda(\hat{y}_r)). \tag{3.22}$$

Through the node $\hat{t}$, it is possible to calculate the robust split $\mathcal{D} = \mathcal{D}_L(\hat{t}, A) \cup \mathcal{D}_R(\hat{t}, A)$ with respect to A. The instances in $\mathcal{D}_L(\hat{t}, A)$ and $\mathcal{D}_R(\hat{t}, A)$ go to the left and right branch respectively. The robust split is calculated as follows:

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

The robust split strategy is the heart of the Treant algorithm. The division of $\mathcal{D}$ in $\mathcal{D}_L(\hat{t}, A)$ and $\mathcal{D}_R(\hat{t}, A)$, guarantees $\mathcal{L}^A$ to be as low as possible with respect to A.

Attack invariance

The attack invariance property preserves the correctness of the robust splitting strategy, each time a node is added to the tree. As explained in 3.5.2 the robust splitting strategy assumes that the attacker is greedy, but it is possible that by adding a new node, new attack points can be created for the attacker. The attacker can choose to alter an instance differently as the leaves of the new node cause a lower loss than a leaf in which it did not fall before. To provide a sound optimization of $\mathcal{L}^A$ on the full dataset $\mathcal{D}$, the attack invariance must be introduced. Let $\mathcal{D}_\lambda \subset \mathcal{D}$, be the set of instances in $\mathcal{D}$ falling in the leaf λ along the tree construction by applying the robust splitting strategy. The security property called attack invariance ensures that the growth step of the tree preserves the correctness of greedy assumptions made on the adversary. Let $\Lambda^A(t,(\boldsymbol{x},y))$ be the set of the leaves of t that are reachable by some attacked instance $\boldsymbol{x}' \in A(\boldsymbol{x})$. The attacker has no advantage in changing the leaves of the set, since they generate the highest loss. The attack invariance ensures that during the construction phase, the set Λ^A contains the leaves of the new split. If this doesn't happen, it means that for the instances in $\mathcal{D}^\lambda$ that reach λ, the attacker found another leaf, outside of Λ^A, that generates a greater loss. This would break the soundness up to the previous step guaranteed by the optimization problem 3.19. Let t' the generated tree by changing a leaf λ of t with a new splitting point $\sigma(f,v,\lambda_l,\lambda_r)$. The tree t' satisfies the attack invariance if and only if:

$$\forall(\boldsymbol{x},y) \in \mathcal{D}^\lambda : \Lambda^A(t',(\boldsymbol{x},y) \cap \{\lambda_l,\lambda_r\} \neq \emptyset. \tag{3.23}$$

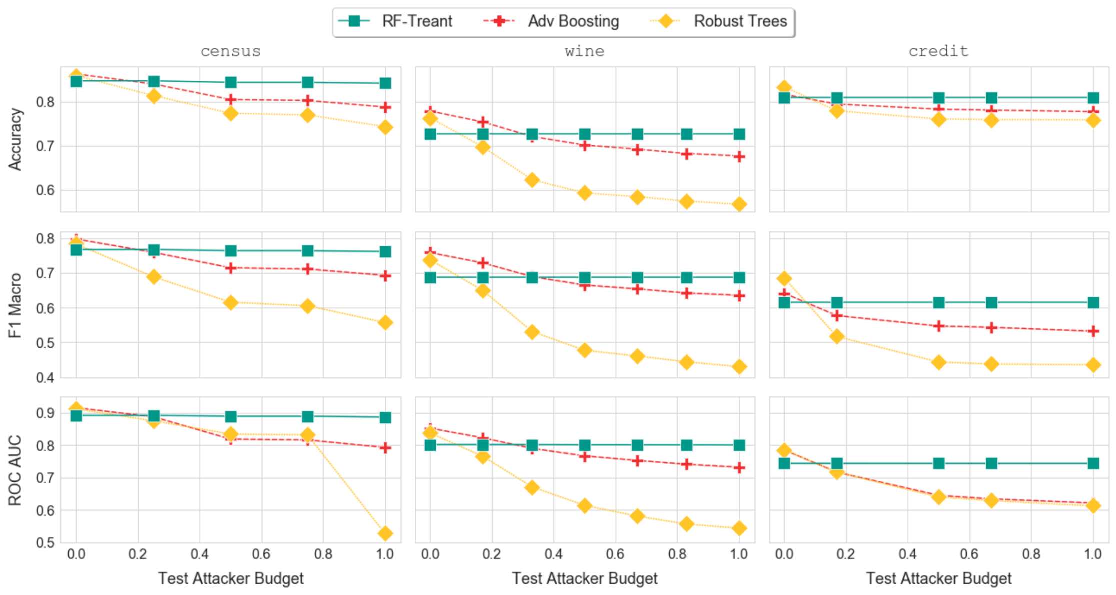
To guarantee the attack invariance, a set constraints are inserted into the optimization problem 3.19. To better understand the concept of constraints, following there is an example taken from [12]. Let $\sigma(f,v,\lambda(\hat{y}_l),\lambda(\hat{y}_r))$ be a new growing node from leaf λ and $\mathcal{D}^\lambda$. Suppose the instance $(\boldsymbol{x},y) \in \mathcal{D}^\lambda$ is sent to the right branch of the node by the robust split, as one of its perturbations $\boldsymbol{x}' \in A(\boldsymbol{x})$ traverses the threshold v and $\ell(\hat{y}_r,y) \geq \ell(\hat{y}_l,y)$. The attack invariance is guaranteed when the leaves $\lambda(\hat{y}_l)$ and $\lambda(\hat{y}_r)$ are respectively replaced with the sub-trees t_l and t_r and there exists an instance $\boldsymbol{x}' \in A(\boldsymbol{x})$ such that the higher loss is generated in a leaf of the sub-tree t_r. If the higher loss for $\boldsymbol{x}'$ is in t_l it means that the growth of the tree has introduced a new point of attack. At this point the attacker can choose to send the instance in the left branch instead in to the right one. To force the attacker to send the instance to the right, the requirement $\ell(\hat{y}_r,y) \geq \ell(\hat{y}_l)$ is transformed into a pair of constraints $\ell(\hat{y}_r,y) \geq \gamma$ and $\gamma \leq \ell(\hat{y}_l)$ where $\gamma = \min\{\ell(\hat{y}_r,y),\ell(\hat{y}_l)\}$. The two constraints are then recursively propagated respectively in the left and right child. As long as the two constraints are not violated, the attacker has no advantage in changing his strategy, so attack invariance is enforced. To implement this constraint mechanism, each leaf λ is extended with a set of constraints $\mathcal{C}$, which is initially empty for the root of the tree. When a leaf λ is split to grow the tree, the constraints are included in the optimization problem 3.19 to determine the best pair $\hat{y}_l, \hat{y}_r$ that preserve the greedy choices made by the attacker so far. Finally the constraints are (partially) propagated to the new leaves and new constraints are generated based on the new split. The constraints guarantee the attack invariably at the price of reducing the space of the possible tree-growing solutions. This property does not prevent the construction of robust decision trees that are also accurate in absence of attacks.

Through these two strategies the Treant algorithm is able to generate a decision tree robust against attacks generated by the attacker's model proposed in 3.5.1, by minimizing the highest loss at each step. Algorithm 6 shows the Treant pseudocode proposed in [12]. Finally in Figures 3.7 we have reported some of the results of the experiments conducted in [12] to compare the performance of a Random Forest of Treant robust trees with Adversarial Boosting (a GBDT trained with adversarial examples) and Robust Tree [13] defined in section 3.4. The experiments were performed on CENSUS, WINE and CREDIT datasets with ACCURACY, F1 MACRO and ROC AUC evaluation metrics [12]. Figure 3.7a shows the performance of the three algorithms keeping the trainig budget fixed and varying the attack budget. From these plots it can be seen that as the attacker's budget grows, Treant constantly outperforms its competitors, especially when the attacker gets stronger. Figure 3.7b shows the performance of the three algorithms keeping the attack budget fixed and varying the training budget. The plots show that Treant still outperforms its competitors and the more the training budget grows, the more Treant improves its performance under attack.

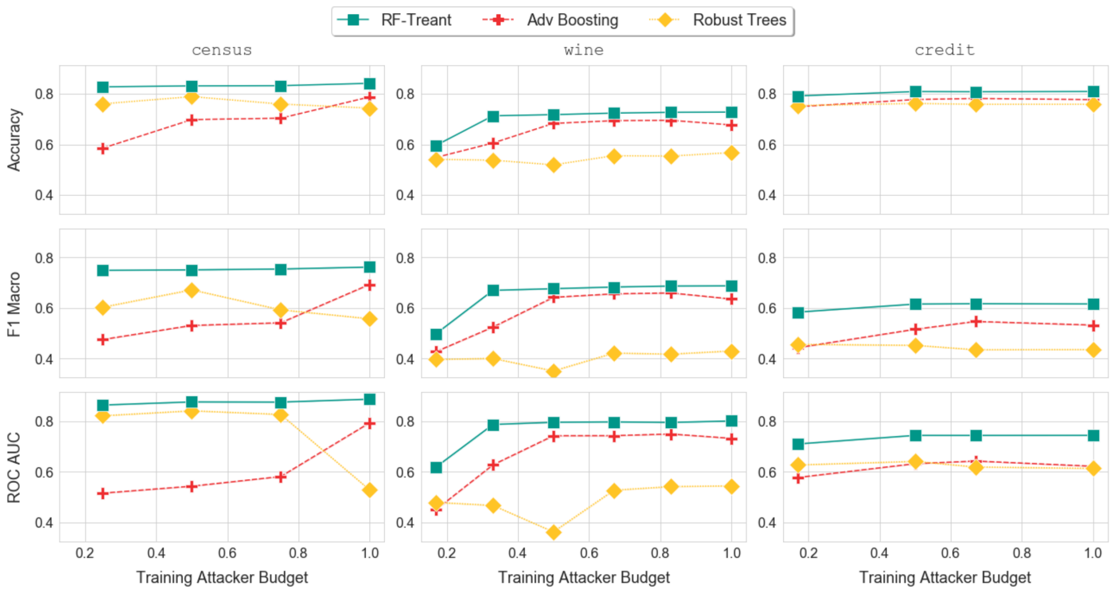
Algorithm 6 TREANT

```
Require: A training set $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_n, y_n)\}$, the attacker model $\mathcal{A}$ and
    the set of constraints $\mathcal{C}$
 1: function Treant($\mathcal{D}, A, \mathcal{C}$)
 2:     $\hat{y} \leftarrow \arg\min_y \mathcal{L}^A(\lambda(y), \mathcal{D})$ subject to $\mathcal{C}$
 3:     $\sigma(f, v, \lambda(\hat{y}_l), \lambda(\hat{y}_r)), \mathcal{D}_l, \mathcal{D}_r, \mathcal{C}_l, \mathcal{C}_r \leftarrow$ TSplit($\mathcal{D}, A, \mathcal{C}$)
 4:     if $\mathcal{L}^A(\sigma(f, v, \lambda(\hat{y}_l), \lambda(\hat{y}_r)), \mathcal{D}) < \mathcal{L}^A(\lambda(\hat{y}, \mathcal{D})$ then
 5:         $t_l \leftarrow$ Treant($\mathcal{D}_l, A, \mathcal{C}_l$)
 6:         $t_r \leftarrow$ Treant($\mathcal{D}_r, A, \mathcal{C}_r$)
 7:         return $\sigma(f, v, t_l, t_r)$
 8:     else
 9:         return $\lambda(\hat{y})$
10:     end if
11: end function

12: function TSplit($\mathcal{D}, A, \mathcal{C}$)
13:     $\mathcal{N} \leftarrow \{\sigma(f, v, \lambda(\hat{y}_l), \lambda(\hat{y}_r)) | f \in [1, d] \land \exists(\boldsymbol{x}, y) \in \mathcal{D} : x^{(f)} = v \ \land \ \hat{y}_l, \hat{y}_r =$
    $\arg\min_{y_l, y_r} \mathcal{L}^A(\sigma(f, v, \lambda(y_l), \lambda(y_r)), \mathcal{D})\}$ subject to $\mathcal{C}$
14:     $\hat{t} = \arg\min_{t \in \mathcal{N}} \mathcal{L}^A(t, \mathcal{D}) = \sigma(f, v, \lambda(\hat{y}_l), \lambda(\hat{y}_r))$
15:     $\mathcal{D}_l \leftarrow \mathcal{D}_L(\hat{t}, A)$
16:     $\mathcal{D}_r \leftarrow \mathcal{D}_R(\hat{t}, A)$
17:     $\mathcal{C}_l \leftarrow \mathcal{C}_L(\hat{t}, A)$
18:     $\mathcal{C}_r \leftarrow \mathcal{C}_R(\hat{t}, A)$
19:     return $\hat{t}, \mathcal{D}_l, \mathcal{D}_r, \mathcal{C}_l, \mathcal{C}_r$
20: end function
```



(a) Comparison of adversarial learning techniques for different test budgets and maximum train budget.



(b) Comparison of adversarial learning techniques for different train budgets and maximum test budget.

Figure 3.7: Results of the analysis on the robustness of the Treant algorithm compared to other robust training techniques. The robustness of the models was compared with ACCURACY, F1 MACRO and ROC AUC, against evasion attacks generated with L_∞-norm. The images of the experiments were taken from [12].

Summary

Briefly, in the first part of this chapter we gave a quick introduction of the state of the art of training techniques to make models as NN, SVM etc., robust to evasion attacks. The second part it focused on the state of the art of learning algorithms and ensemble methods based on decision trees, robust against evasion attacks. The chapter highlighted what has been done so far on ensemble methods and decison tree against evasion attacks. The models analyzed in this chapter are the following:

- **Boosted Decision Tree - Robust.** The BDT-R model is an ensemble methods based on Boosted Decision Tree and is trained through adversarial training. In the article they introduced two strategies for creating evasion attacks, one called **optimal evasion** based on a very expensive but precise **MILP** problem and one based on a faster but less accurate technique called **symbolic prediction**. Through symbolic prediction they trained a BDT-R on evasion attacks with distance L_0-norm less than the attacker's budget b
- **Random Subspace Method.** RSM is an ensemble methods designed to increase the accuracy of individual base-learners. It trains each base-learner on a restricted subset of randomly sampled features. Later, in other researches, this construction was exploited to better distribute the most discriminating features among the various base-learners and increase robustness against adversarial examples bound by distance function L_0. In the experiments we compared our model with RSM.
- **Robust Split.** The Robust Split algorithm bases its robustness by maximizing the minimum score under attack. In other words, it divides the instances so as to generate the highest score when the model is under attack. The algorithm trains the model based on attacks generated by an adversary constrained by distance function L_∞. The algorithm was presented in two versions. The first based on maximizing the minimum score under attack calculated by information gain score. While the second version is based on GBDT and the minimum score under attack is calculated through four specific cases of distribution of instances with respect to the left or right branch of the tree.
- **Treant.** The Treant algorithm trains a robust decision tree to attacks generated by an adversary whose strength is constrained by distance function L_∞. The robustness of the tree is guaranteed through two properties: the **robust splitting** which divides the training data in order to generate the lowest maximum loss under attack and the **attack invariance** which allows to maintain the assumptions made during the robust splitting consistent with the growth of the tree.

Chapter 4

Feature Partitioned Forest

This chapter represents the heart of this thesis, as well as the scientific contribution that we have given regarding adversarial machine learning. This chapter introduces our ensemble learning algorithm called Feature Partitioned Forest (FPF) based on decison trees and robust to evasion attacks. The chapter also contains the definitions of the two certificates to calculate the performance of FPF without performing all the possible attacks.

4.1 Adversary's Model

Before giving the definition of our algorithm, it is necessary to specify the attacker's model for which it was designed. The attacker's model is defined on *evasion attacks*, where an attacker aims at fooling an already trained classifier by maliciously modifying a given instance before submitting it to the classification model. The perturbation caused by the attacker is not unconstrained as the attack should be *"invisible"* to the classification system. As in Kantchelian [32], we assume an attacker A_b that is capable of modifying a given instance $\boldsymbol{x}$ into a perturbed instance $\boldsymbol{x}'$ such that the L_0-norm of the perturbation is smaller than the attacker's budget k, i.e., $\|\boldsymbol{x} - \boldsymbol{x}'\|_0 \leq b$ with $b \in \mathbb{N}^+$. Therefore, attacker A_b can perturb the instance $\boldsymbol{x}$ by modifying at most b features, without any constraint on how much a given feature can be altered. Indeed, a very small b is sufficient to achieve successful attacks. Su [44] show that with a one-pixel attack, i.e., with $b = 1$, it is possible to fool a complex deep neural network as VGG16 [42] and decrease its accuracy to a poor 16%. Given an instance $\boldsymbol{x} \in \mathcal{X}$, we denote by $A_b(\boldsymbol{x})$ the set of all the perturbed instances the attacker may generate:

$$A_b(\boldsymbol{x}) = \{\boldsymbol{x}' \mid \boldsymbol{x}' \in \mathcal{X} \ \wedge \ \|\boldsymbol{x} - \boldsymbol{x}'\|_0 \leq b\}\,. \tag{4.1}$$

The power of the attacker that we have proposed is bound by the value of b. The budget can take a value between $0 \leq b \leq \lceil d/2 \rceil - 1$ [1]. In other words, attacker A_b can perturb the instance $\boldsymbol{x}$ by modifying at most b features, and there are no limits on how much perturbed features can be changed.

4.2 Model definition

before continuing with the definition of the FPF model, it is essential to introduce the concept of features partitioning and when a partitioning can be considered robust against attacks produced by an adversary A_b.

4.2.1 Feature Partitioning

Given $\mathcal{P}$ a *set partition* of the feature set $\mathcal{F}$ and an attacker A_b that decided to corrupt the set of features $B \subseteq \mathcal{F}$, with $|B| \leq b$, It is easy to compute the number of sets in $\mathcal{P}$ overlapping with B as:

$$O(\mathcal{P}, B) = \sum_{P \in \mathcal{P}} \mathbb{1}[B \cap P \neq \emptyset] \tag{4.2}$$

where $\mathbb{1}[e]$ equals 1 if expression e is true and 0 otherwise. Partition $\mathcal{P}$ is called *robust* if the majority of its sets cannot be impacted by the attacker A_b, i.e., if the following property holds:

$$\forall B \subseteq \mathcal{F}, |B| \leq b \quad O(\mathcal{P}, B) < \frac{|\mathcal{P}|}{2}. \tag{4.3}$$

When $|B| \leq b$, it is straightforward to show that this property is surely satisfied if $|\mathcal{P}| \geq 2b + 1$. Consider the worst case: at most b distinct subsets of $\mathcal{P}$ can have an overlap with B, leaving other $\geq b + 1$ subset of $\mathcal{P}$ unaffected. Hereinafter, we consider only *robust* feature partitions $\mathcal{P}$ where $|\mathcal{P}| = 2b + 1$.

4.2.2 Robust Forest

Given the dataset $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ with $\boldsymbol{x} \in \mathcal{X}^d$ and $y \in \{-1, 1\}$, FPF trains a forest of binary decision trees and the resulting ensemble is a binary classifier. The robustness of the ensemble method against evasion attacks generated by A_b is implemented as follows. Let's consider a forest $\mathcal{T}$ that, given an attacker A_b, is built by exploiting a robust feature partition $\mathcal{P}$ as follows. Let $\mathcal{D}$ be a set of training instances $\boldsymbol{x} \in \mathcal{X}$, and $\mathcal{P}$ be a robust partition of its feature space $\mathcal{F}$. Given

[1]The reason for this constraint is explained in the **Observations** section 4.4

$P \in \mathcal{P}$, we call $\pi_P(\mathcal{D})$ the projection of $\mathcal{D}$ on the feature set P, i.e., the dataset obtained from $\mathcal{D}$ by discarding those features not included in P. Given a robust feature partitioning $\mathcal{P}$, it is thus possible to build a robust forest by training $2b+1$ trees independently on the $2b+1$ projections $\pi_P(\mathcal{D})$, with $P \in \mathcal{P}$. The algorithm sketched above achieves what we formally define as *robustness.*

Definition 1 (Robust Forest). *Given an attacker A_b, a forest $\mathcal{T}$ is* robust *if the majority of its trees is not affected by A_b for any of its attacks:*

$$\forall \boldsymbol{x} \in \mathcal{X}, \forall \boldsymbol{x}' \in A_b(\boldsymbol{x}) \quad \sum_{t \in \mathcal{T}} (\mathbb{1}[t(\boldsymbol{x}) = t(\boldsymbol{x}')]) > \frac{|\mathcal{T}|}{2} \tag{4.4}$$

It is straightforward to show that if a forest is built on the basis of a robust feature partitioning $\mathcal{P}$ as described above, then, at most b of its $2b+1$ trees can be affected by the attacker. Note that, the above definition and training strategy trivially generalizes to any ensemble learning algorithm.

Increasing the accuracy of a robust forest. The above definition does not provide any guarantee on the accuracy of the full forest $\mathcal{T}$, which clearly depends on the accuracy of its single trees. Yet, the more accurate the trees $t \in \mathcal{T}$, the more likely the forest $\mathcal{T}$ is accurate under attack. One limitation of feature partitioning is that each single tree is trained on a reduced number of features. In this regard, to increase the accuracy of a robust forest $\mathcal{T}$, we equi-partition $\mathcal{F}$ across $\mathcal{P}$ so as to have $|P| = \lfloor \frac{|d|}{2b+1} \rfloor$ for all $P \in \mathcal{P}$. Clearly, as the attacker's power b increases, we require to partition $\mathcal{F}$ in to a larger number of subsets, and for these to be effective we require the dataset to have a larger number of high quality features. Note that this is true for every learning algorithm: if the attacker can perturb at will b features, we require to have more than b high quality features to train an accurate model. The ideal case is the perfect accuracy of $\mathcal{T}$, which guarantees that $\mathcal{T}$ is resilient *by construction* to an attacker A_b. Suppose that for a given test instance $(\boldsymbol{x}, y)$, we have that $\forall t \in \mathcal{T}$, $t(\boldsymbol{x})y > 0$: in this case attacker A_b does not have any chance to modify the prediction of $\mathcal{T}$, i.e., $\forall \boldsymbol{x}' \in A_b(\boldsymbol{x})$, $\mathcal{T}(\boldsymbol{x}) = \mathcal{T}(\boldsymbol{x}')$ by construction. In fact, the *maximal attack* to $\mathcal{T}$, which can break b trees by perturbing b features, cannot however modify the prediction of the majority of the $2b+1$ trees in $\mathcal{T}$.

Increasing the robustness of a robust forest. In the ideal case, our model can never be circumvented by an attacker A_b. However in the real case since we use a limited number of features for each tree, some of them may not be perfectly accurate. This fact undermines the robustness by construction of the FPF algorithm. Specifically, given a test instance $(\boldsymbol{x}, y)$, even if $\mathcal{T}(\boldsymbol{x})y > 0$, individual trees may still generate incorrect results. For example, if $\exists t \in \mathcal{T},\ t(\boldsymbol{x})y < 0$, let $\boldsymbol{x}' \in A_b(\boldsymbol{x})$ be an evading instance obtained by perturbing b features aimed to break b (otherwise accurate) trees. This attack possibly succeeds in misleading $\mathcal{T}$, i.e., $y \cdot \mathcal{T}(\boldsymbol{x}') < 0$, despite our robust forest construction. Furthermore, the trees inside the ensemble are built on different features, so they can present a different strength to the attacks. For example a tree could break with only one feature attacked, while for another one would need more. From these last observations an attacker can maximize the damage by deciding how to spend his budget by exploiting the independent errors committed by the forest and the robustness of the single trees inside it. To overcome this problem we decided to increase the size of the forest $\mathcal{T}$. Indeed, we generate an ensemble of r forests $\mathcal{T} = \{\mathcal{T}_i\}_{i=1,\dots,r}$, where each single $\mathcal{T}_i$ is always composed of $2b+1$ trees, but is build upon a different and randomly selected equi-partition $\mathcal{P}_i$ of $\mathcal{F}$. Using multiple rounds of the algorithm brings several advantages. First of all, multiple rounds r increases the difference between attackable and non-attackable trees. This increase allows to overcome the problem of independent errors made by trees inside the forest. For example, let's consider a forest trained with parameters $b = 1$ and $r = 1$, the ensemble has a size of 3 trees. In the worst case the attacker breaks at most 1 tree. If at least one of the remaining 2 trees produces a wrong prediction of $\boldsymbol{x}'$ (or $\boldsymbol{x}$, is the same) then the prediction of the ensemble $\mathcal{T}$ will be wrong. In this case, an independent error is sufficient to break the model. On the other hand, if $r = 100$ the forest contains 300 trees, of which at most 100 can be attacked by the adversary and at least 200 cannot be attacked. In this case, at most 49 out of 200 trees can break independently to still ensure the correct prediction of the ensemble. So increasing r introduces $\lceil \frac{rb}{2} \rceil - 1$ trees of protection against independent failures. Furthermore, the repetition of the construction, allows to have many more base-learners and consequently a more heterogeneous ensemble. In this way the attacker has more difficulty in finding a combination of b features that breaks all the trees contain them. It is very important to note that even if there are trees with common features between different rounds, the robust partition property is maintained because every $\mathcal{T}_i \in \mathcal{T}$ is defined on a robust partition. The proof of this statement is given in 4.4. Finally, another advantage of using multiple algorithm rounds is that at each round, the features of each tree are sampled randomly. This randomness of the distribution of fatures within the base-learners indirectly exploits the same idea of RSM 3.3 in implementing the robustness strategy proposed by [33]. Briefly in [33] they said that separating discriminative features from less discriminative ones allows the models to discover new important features that were previously eclipsed. The consequence is that an adversary has to modify a greater number of features to evade the model [5]. Finally in 7 we reported the pseudocode of the FPF algorithm to train a forest $\mathcal{T}$ aimed to be robust against an attacker

A_b that can perturb at most b features. The algorithm takes as input a dataset $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, with $\boldsymbol{x} \in \mathcal{X} \subseteq \mathbb{R}^d$, the features set $\mathcal{F}$, the attacker budget b and the number of round r. During each round $i = 1, \ldots, r$, a new forest $\mathcal{T}_i$ is added to $\mathcal{T}$, where $|\mathcal{T}_i| = 2b+1$. Each $\mathcal{T}_i$ is trained on a different and randomly chosen robust partition $\mathcal{P}_i$ of $\mathcal{F}$. Specifically, at each round the features set $\mathcal{F}$ is shuffled and then equi-split into $2b+1$ disjoint chunks, used to project $\mathcal{D}$ so that each $t \in \mathcal{T}_i$ is trained with only the features specified in one of the chunks.

Algorithm 7 Robust_Partitioning_of_Features

```
Require: A training set $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, the features space $\mathcal{F}$, the
    attacker budget $b$ and the training rounds $r$
 1: function Robust_Partitioning_of_Features($\mathcal{D}, \mathcal{F}, b, r$)
 2:     $\mathcal{T} \leftarrow \emptyset$
 3:     $k \leftarrow 2 \cdot b + 1$
 4:     for $i = 1$ to $r$ do
 5:         $\mathcal{T}_i \leftarrow \emptyset$
 6:         $\mathcal{P}_i \leftarrow \mathsf{RobustPartition}(\mathcal{F}, k)$
 7:         for all $P_j \in \mathcal{P}_i, j = 1, \cdots, k$ do
 8:             $t \leftarrow \mathsf{DecisionTree}(\pi_{P_j}(\mathcal{D}))$
 9:             $\mathcal{T}_i \leftarrow \mathcal{T}_i \cup t$
10:         end for
11:         $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{T}_i$
12:     end for
13:     return $\mathcal{T}$
14: end function
```

4.3 Certificates of robustness

In this section we propose two certificates of robustness that allow to calculate a lower bound of the performance of the model under attack without computing all the possible evading instances in $A_b(\boldsymbol{x})$. The first lower bound is called *fast lower-bound*, whose name derives from its polynomial complexity. The second is called *accurate lower-bound*, which has greater accuracy than the first, but at an exponential cost. Despite the exponentiality of the accurate lower-bound, it is extremely faster than computing $A_b(\boldsymbol{x})$. It is important to note that the two proposed lower-bounds allow to compute the robustness of a model with respect to an instance $\boldsymbol{x}$ and an attacker A_b. The certificates answer *"yes"* or *"I don't know"* to the question *"is the model robust against an attack of A_b on $\boldsymbol{x}$?"*. If the certificate answers *"yes"* then for certification there is no perturbation of $\boldsymbol{x}$ on b

features that eludes the model, $\forall \boldsymbol{x'} \in A_b(\boldsymbol{x}) : \mathcal{T}(\boldsymbol{x'})y > 0$. If the answer is *"I don't know"*, there is no guarantee of the existence of the attack. The uncertainty of the certificate is a consequence of the assumptions made for the approximation. Table 4.1 shows some useful definitions for the explanation of each certificate.

Table 4.1: Certificate notation.

Notation	Definition
$\mathcal{F}_b^{(i)}$	$\mathcal{F}_b^{(i)} \subseteq \mathcal{F}$ with $\lvert\mathcal{F}_b^{(i)}\rvert = b$.
$\mathcal{F}_b$	$\{\mathcal{F}_b^{(i)} \mid \forall \mathcal{F}_b^{(i)} \subseteq \mathcal{F}\}$.
$\mathcal{F}_{\boldsymbol{x}}^t$	set of features of $\boldsymbol{x}$, tested in the decision path of $t(\boldsymbol{x})$.
$\mathcal{T}_{\boldsymbol{x}}^f$	set of trees, which feature f is tested in the prediction of $\boldsymbol{x}$
$\mathcal{T}_i^A$	$\{t \mid \forall t \in \mathcal{T}, \mathcal{F}_{\boldsymbol{x}}^t \cap \mathcal{F}_b^{(i)} \neq \emptyset\}$
$\boldsymbol{a_x}$	$a_{\boldsymbol{x}}^{(f)} = \sum_{t \in \mathcal{T}_{\boldsymbol{x}}^f} \mathbb{1}[t(\boldsymbol{x})y > 0], \ \forall f \in [1, d]$
$\bar{\boldsymbol{a}}_{\boldsymbol{x}}$	vector $\boldsymbol{a_x}$ sorted in descending order

4.3.1 Fast lower-bound: FLB

Fast lower-bound (FLB) is an algorithm that looks for the b features inside the ensemble that cause the greatest damage. The approximation is based on the pessimistic assumption that if a tree uses an attacked features to predict the label of an instance $\boldsymbol{x}$, then the instance will be classified incorrectly. Therefore the approximation underestimates the strength of the individual tree under attack. To deal with the explanation of the lower-bounds it is necessary to know the meaning of *prediction path set.* The prediction path set denoted as $\mathcal{F}_{\boldsymbol{x}}^t$, is the set of features used by $t \in \mathcal{T}$ when predicting $\boldsymbol{x}$. Given the instance $\boldsymbol{x}$, the decision path set is created for each tree inside the forest. This set allows to understand which features must be attacked in order to break t. Subsequently, through the decision path sets, for each features in $f \in [1, d]$, the algorithm creates the *attacked tree set* denoted as $\mathcal{T}_{\boldsymbol{x}}^f$, each of which represents the set of potentially breakable trees by attacking the feature f. Through these sets it is possible to calculate the *damage vector* $\boldsymbol{a_x}$, formally defined in Table 4.1. Each element $a_{\boldsymbol{x}}^{(f)} \in \boldsymbol{a_x}$ is associated with a feature and maintains the maximum number of trees that the attacker can advantageously break by attacking the feature f. Attacking with advantage means attacking a tree that correctly predicts the label of $\boldsymbol{x}$. The attacker has no advantage in wasting budgets to attack an already broken tree. The vector $\boldsymbol{a_x}$ is sorted in decreasing order of damage and it is denoted as $\bar{\boldsymbol{a}}_{\boldsymbol{x}}$. In this way the first b elements represent the features that break the largest number of trees. The vector $\bar{\boldsymbol{a}}_{\boldsymbol{x}}$ allows the algorithm to approximate the adversary's strongest attack. To check if the model is robust to an attack on b features, the algorithm computes following formula:

$$o_{\boldsymbol{x}}^{\text{FLB}} = y \sum_{t \in \mathcal{T}} t(\boldsymbol{x}) - \sum_{i=1}^{b} 2(\bar{\boldsymbol{a}}_{\boldsymbol{x}})^{(i)} \tag{4.5}$$

If $o_{\boldsymbol{x}}^{\text{FLB}}$ is greater than zero, then the model correctly classifies each evading instance in $A_b(\boldsymbol{x})$. The reason for the multiplication by 2 inside the rightmost sum is given by the fact that in the first $\sum_{t \in \mathcal{T}} t(\boldsymbol{x})y$ all the trees are counted, even those attacked. In this way, positive predictions are subtracted and errors are marked. In this lower bound the approximation is given by two assumptions. The first as mentioned above, if a tree is attacked even on a single feature of the prediction path set of $\boldsymbol{x}$, it is considered broken. The second assumption is not to consider relevant the fact that two or more features in $\bar{\boldsymbol{a}}^{\boldsymbol{x}}$ can share the same tree and therefore it would be counted more than once. To avoid this last problem, it is necessary to generate all the $C_{(d,b)}$ combinations of features and count attacked trees correctly. The latter consideration is used in the accurate lower-bound.

Fast lower-bound complexity To calculate the temporal complexity of FLB it is necessary to analyze the cost of creating each data structure. By definition $|\mathcal{T}| = r(2b+1)$, so the dimension of the set of trees is $O(rb)$. The cost of scanning the set of features $\mathcal{F}$ is $O(d)$. Again by construction each features appears in a tree per round then $|\mathcal{T}_{\boldsymbol{x}}^{f}| = r$. To calculate the decision path set $\mathcal{F}_{\boldsymbol{x}}^{t}$ it is necessary to traverse t from the root to the leaf corresponding to the prediction of $\boldsymbol{x}$. In a balanced binary tree the height h of the tree is given by $\log_2(l-1)$ where l is the number of leaves of the tree. Since each tree can have a different number of leaves, we refer to h as the height of the tree with the largest number of leaves. The cost of the prediction $t(\boldsymbol{x})$ is therefore $O(\log_2(l)) = O(h)$. So the cost of calculating $\mathcal{F}_{\boldsymbol{x}}^{t}$, $\forall t \in \mathcal{T}$ is $O(bhr)$. By construction the features in $\mathcal{F}$ are distributed in $2b+1$ trees per round, so each tree has at most $|\mathcal{F}_{\boldsymbol{x}}^{t}| = \frac{d}{2b+1}$ features. The cost of scanning $\mathcal{F}_{\boldsymbol{x}}^{t}$ is therefore $O(\frac{d}{b})$. The resulting sets $\mathcal{T}_{\boldsymbol{x}}^{f}$, $\forall f \in [1, d]$ The resulting sets with respect to the sets $\mathcal{F}_{\boldsymbol{x}}^{t}$, $\forall t \in \mathcal{T}$. Through a dictionary that maps each feature to a list of trees, it is possible to create the inverted index with the following computational complexity. To create this inverted index, Marco visits each feature $f \in \mathcal{F}_{\boldsymbol{x}}^{t}$ at the cost of $O(\frac{d}{b})$. The cost of visiting each features sets $\mathcal{F}_{\boldsymbol{x}}^{t}$ for each tree $t \in \mathcal{T}$ is $O(\frac{d}{b} \cdot br) = O(dr)$. When a tree is added to a feature list, there is no need to check that it is already there because $\mathcal{F}_{\boldsymbol{x}}^{t}$ is a set and never has two identical features associated with the same tree. If we assume that the predictions $t(\boldsymbol{x}), \forall t \in \mathcal{T}$ obtained during the calculation of the decison path sets are saved, the cost of computing an element $a_{\boldsymbol{x}}^{(f)}$ of the damage vector $\boldsymbol{a}_{\boldsymbol{x}}$ will be $O(r)$ as it is sufficient visit the set $\mathcal{T}_{\boldsymbol{x}}^{f}$ and count the correct predictions. The size of the damege vector is $|a| = |\mathcal{F}| = d$. For each element of $\boldsymbol{a_x}$ the temporal complexity is $O(dr)$. Finally, the damage vector $\bar{\boldsymbol{a}}_{\boldsymbol{x}}$ is sorted in decreasing order with a sorting cost of $O(d\log(d))$. The temporal complexity for creating data structures is $O(bhr + dr + d\log(d)) = O(bhr + dr + d\log(d))$. With the prediction of the trees saved, the complexity of the two summations is $O(rb)$. The final temporal complexity of fast lower-bound is $O(bhr + dr + d\log(d))$. The temporal complexity depends on the training parameters r and b, the size of the feature space d and the height of the trees h.

4.3.2 Accurate lower-bound: ALB

Unlike the FLB, accurate lower-bound (ALB) algorithm only makes one assumption that defines its approximation: the classification made by any tree that uses a feature under attack in the prediction of $\boldsymbol{x}$ is considered wrong. For each combination of d features taken b at a time $\mathcal{F}_b^{(i)} \in \mathcal{F}_b$, the algorithm calculates the set of attackable trees $\mathcal{T}_i^A$ defined as in 4.1 and the set of safe trees $\mathcal{T}_i^S = \mathcal{T} \setminus \mathcal{T}_i^A$. Trees in $\mathcal{T}_i^A$ are those that have features of their decision path set $\mathcal{F}_{\boldsymbol{x}}^t$ in common with $\mathcal{F}_b^{(i)}$, and they are therefore attacked. The accurate lower-bound for $\boldsymbol{x}$ with respect to an attacker A_b is calculated using the following formula:

$$o_{\boldsymbol{x}}^{\text{ALB}} = \prod_{i=1}^{C_{(d,b)}} \mathbb{1}\left[\left(y \sum_{t\in\mathcal{T}_i^S} t(\boldsymbol{x}) - |\mathcal{T}_i^A|\right) > 0\right] \tag{4.6}$$

If $o_{\boldsymbol{x}}^{\text{ALB}}$ is 1 it means that there is no parturbation $\boldsymbol{x}' \in A_b(\boldsymbol{x})$ which produces $t(\boldsymbol{x})y < 0$. If the difference $y\sum_{t\in\mathcal{T}_i^S} t(\boldsymbol{x}) - |\mathcal{T}_i^A|$ is positive then it means the majority of the forest predicts the correct label and the combination of features $F_b^{(i)}$ is not an attack. If the difference is positive, the $\mathbb{1}$ function returns 1, 0 otherwise. Through the result of $\mathbb{1}$, the product returns 0 if there is at least a combination of features $\mathcal{F}_b^{(i)}$ which causes an error. Since the algorithm must test all possible combinations of the d features taken b at a time, this approximation is much more expensive than the previous ones, but at the same time it is more accurate since trees are counted only once.

Accurate lower-bound complexity Also in this case, to calculate the temporal complexity of the algorithm, it is necessary to divide the analysis into pieces. From above, the complexity for creating all the decision path sets is $O(bhr)$. As can be seen from the definition of $\mathcal{T}_i^A$ in table 4.1 for each $t \in \mathcal{T}$ (the scanning cost of the forest is $O(br)$), ALB computes the intersection $\mathcal{F}_{\boldsymbol{x}}^t \cap \mathcal{F}_b^{(i)}$. If we assume that the sets $\mathcal{F}_{\boldsymbol{x}}^t$ and $\mathcal{F}_b^{(i)}$ are ordered then the intersection costs as the length of the longest list:

$O(\max\{\frac{d}{b}, b\})$. Since the sorting of the sets can be done a priori outside the product, it is not involved in the search for all combinations. Therefore the costs of ordering each $\mathcal{F}_{\boldsymbol{x}}^t$ is $O(\frac{d}{b}\log(\frac{d}{b}))$, for each tree inside the forest the temporal complexity of sorting all the sets is $O(dr\log(\frac{d}{b}))$. Furthermore, the algorithm that we used to generate all the combination sets $\mathcal{F}_b^{(i)} \in \mathcal{F}_b$, creates them already ordered, therefore we can omit the temporal complexity $O(b\log(b))$. The final cost of creating $\mathcal{T}_i^A$ is $O(br \cdot \max\{\frac{d}{b}, b\})$. If the predictions $t(\boldsymbol{x}), \forall t \in \mathcal{T}$ are saved when calculating the $\mathcal{F}_t^{\boldsymbol{x}}$, the complexity of $\sum_{t\in\mathcal{T}_i^S} t(\boldsymbol{x})$ is $O(br)$ which is less than $O(br \cdot \max\{\frac{d}{b}, b\})$ and therefore can be omitted. The final temporal complexity of accurate lower-bound is given by the cost of creating $\mathcal{T}_i^A$ for each combination, added to the cost of calculating and sorting each $\mathcal{F}_{\boldsymbol{x}}^t$, so it is $O(C_{(d,b)}br \cdot \max\{\frac{d}{b}, b\} + bhr + dr\log(\frac{d}{b}))$. The temporal complexity of accurate lower-bound grows exponentially with the growth of b and d, due to the number of combinations of attacks.

Due to the exponential growth of the combinations, the accurate lower-bound can be used to give the robustness of a model for a limited attack strength. On the contrary, as the name suggests, fast lower-bound prefers speed of execution over accuracy. This lower-bound can be used to perform a deep analysis on the model, especially when analyzing the robustness of the model as the attacker's strength increases. Or it can be used to find the best trainig parameter b without performing all the attacks and checking the real robustness of the model. It is also possible to use the two certificates in a combined way when tuning the model parameters. In fact, the fast lower bound can be used to reduce the space of possible values. Once a small set of good values has been found for the analyzed parameters, it is possible to verify them with the accurate lower-bound for greater accuracy. Finally, as explained in more detail in the next chapter, fast lower-bound can be used to significantly reduce the search for possible attacks, excluding those instances that cannot be attacked.

4.4 Observations

Limit of robustness: FPF algorithm has a limitation on the maximum number of features it can protect. In fact the training parameter b must be $b \leq \lceil d/2 \rceil - 1$.

Proof. Suppose for absurdity to construct a model that protects $b \geq \lceil d/2 \rceil$ features. By definition of the model, the ensemble has a number of trees equal to $2b + 1 \geq 2 \cdot \lceil d/2 \rceil + 1$, each with features distributed with robust partitioning. There are two cases:

- if d is even then $2b + 1 \geq 2 \cdot \lceil d/2 \rceil + 1 = d + 1 > d$
- if d is odd instead $2b + 1 \geq 2 \cdot \lceil d/2 \rceil + 1 = d + 2 > d$

In both cases the size of the ensemble is larger than the number of features. This is absurd by the definition of the ensemble. It is not possible to distribute d features on a larger number of trees, unless for some empty trees or to repeat some features. $\square$

This limit is not so problematic. For example, if an attacker can modify all the features of a sample $\boldsymbol{x}$ as much as he wants, it is not possible in any way to create a useful model capable of predicting the correct label of the instance. This because the attacker could transform an instance $\boldsymbol{x}$ from class y into any instance of another class $y' \neq y$. So limiting the attacker's strength to $b \leq \lceil d/2 \rceil - 1$ is reasonable.

Robustness preserved: In this section, we have demonstrated that even for a number of rounds $r > 1$, an attacker A_b can attack only less than half the forest. So the robustness property guaranteed by robust partitioning is preserved.

Proof. Given the number of features to be protected $b \in \mathbb{N}^+$ and the number of rounds $r \in \mathbb{N}^+$, the algorithm creates an ensemble of $r(2b+1)$ trees. The minimum number of trees to have the majority in the prediction is $\lfloor r(2b+1)/2 \rfloor + 1$. By definition, FPF creates r forests of $2b+1$ trees with robust partitioning. By construction only b trees for each round are attackable by A_b. Consequently, A_b can attack at most br trees in the ensemble. Given $r \in \mathbb{N}^+$ we can say that $\lfloor r(2b+1)/2 \rfloor + 1 = \lfloor (2br+r)/2 \rfloor + 1 = br + \lfloor r/2 \rfloor + 1 \geq br + 1 > br$, then the number of attackable trees is always less than the number of non-attackable trees. $\square$

Possible attack scenario: Suppose a context in which an attacker can attack b features taken from a small subset of features $\mathcal{F}^A \subset \mathcal{F}$. Is it useful to remove these features from the model to make it safe? No, the removal of the features leads to a decrease in the accuracy of the model. A model designed to be robust should maintain performance similar to a non-robust model on normal instances. In this context, the FPF model allows to maintain the features of the subset $\mathcal{F}^A$ and guarantee the robustness of the model. Also, the more $\mathcal{F}^A$ grows, the more features are removed from the datast. If $\mathcal{F}^A = \mathcal{F}$ all features are potentially attackable, therefore they should all be removed and therefore it would be impossible to generate a model.

Summary

This chapter is the most important of the thesis because we have presented our ensemble learning algorithm called **Feature Partitioned Forest**. FPF is an ensemble method based on decison tree robust to evasion attacks. The chapter can be summarized in the following main points:

- **Attacker's model.** In the first section of this chapter we presented the attacker model for which FPF must be robust and on which we have performed the experiments. The attacker denoted as A_b can perturb each instance with respect to the distance function L_0 with at most b changes.

- **Model definition.** FPF bases its robustness on a **robust** feature partition that divides the dataset features into $2b + 1$ sets (robust = each partition doesn't share features). Each $2b+1$ sets is used to train a base-learner. Given the attacker A_b, through this robust partitioning it is possible to guarantee that the largest number of base-learners within the ensemble is not involved in the attack. We subsequently showed that aggregating multiple groups of $2b+1$ base-learners, trained upon a different and randomly selected robust partition of the features set, leads to an increase in the robustness of the model. The final size of the ensemble produced by FPF is of $r(2b + 1)$ base-learners.

- **Certificates.** Together with the model, we presented two certificates of robustness based on the structure of the decision trees: **fast lower-bound** and **accurate lower-bound** which respectively allow a fast and accurate calculation of the lower bound of the performance of a model under attack, without performing all possible attacks. Furthermore, for each certificate we calculated the temporal complexity.

- **Observations.** In the last section of this chapter we have reported some observations and demonstrations about FPF. For examples we demonstrated that increasing the size of the ensemble with multiple groups of $2b + 1$ trees does not compromise the robustness by construction. Or we demonstrated that the model has a construction constraint on the maximum number of features it can protect.

Chapter 5

Evaluation Methodology

This chapter details how we performed the experiments and evaluations to compare the robustness of Feature Partitioned Forest with state-of-the-art approaches. The first part of the chapter concerns the experimental settings, where we introduced the datasets used, the models compared and the strength of the attacker A_b used for the evaluation of robustness. Next, we defined the *accuracy under attack*, a robustness evaluation metric and then we presented the brute-force algorithm for generating evasion attacks. For some training parameters, we have defined targeted experiments to perform the best tuning in favor of the robustness of each model. In this way the comparison between the models was as fair as possible. Finally, we have defined some experiments to demonstrate that the accuracy under attacks, approximated by the two lower bounds, is very close to the real value calculated by our brute-force algorithm.

5.1 Experimental settings

The purpose of this section is to detail the datasets used and their characteristics, the models used in the experiments and their training parameters and finally the definition of the strength of the attackers used. For each dataset, we gave a description of what it represents and the possible purpose of an adversary attacking an instance inside it. Subsequently we introduced the algorithms used in the comparisons and for each one we described the training parameters that we analyzed in the experiments. Finally, from the definition of attacker's model given in section 4.1, we have specified the possible budgets of the attacker used during the experiments.

Table 5.1: Datasets description.

dataset name	$\|\mathcal{F}\|$	**#imp**	**maj. class**	$\|\mathcal{D}\|$	$\|\mathcal{D}_{train}\|$	$\|\mathcal{D}_{test}\|$
Spam_Base	57	26	60.6%	4600	3036	1564
Breast_Cancer	30	15	62.7%	569	375	194
Binary_Wine	13	7	73.0%	178	117	61
MNIST 0/1	784	54	53.3%	14780	9754	5026
MNIST 1/7	784	79	51.9%	15170	10012	5158
MNIST 5/6	784	173	52.1%	13189	8704	4485

5.1.1 Datasets

In the experiments we used the following datasets: **Spam_Base**, **Breast_Cancer**, **Binary_Wine**, **MNIST 0/1**, **MNIST 1/7** and **MNIST 5/6**. The datasets characteristics are summarized in Table 5.1. In order, the fields in the table represent: $|\mathcal{F}|$ the cardinality of the features space of the dataset, **#imp** indicates the number of important features within the datasets. This value was calculated by training a Random Forest with 100 estimators and counting the features whose collective importance represents 90% of the total. The **maj. class** (majority class) represents the percentage of the majority class within the dataset. The whole dataset is noted with $\mathcal{D}$. The instances in $\mathcal{D}$ were divided into: 0.66% for the training set $\mathcal{D}_{train}$ and 0.34% for the test set $\mathcal{D}_{test}$. The division of the instances into the two sets was done with stratified sampling to maintain the proportion of the classes. Hereinafter it must be considered that each model was trained with instances in $\mathcal{D}_{train}$ and evaluated on set $\mathcal{D}_{test}$. Each value reported in the experiment results is based on the instances in $\mathcal{D}_{test}$. The description of the meaning of each dataset is given below.

- **Spam Base** [19] is an email classification dataset representing spam / ham messages. Each entry in the dataset is a vector of 57 features. The first 54 are the percentage of 48 higly discriminative words and 6 characters, calculated through baf-of-words. Feature 55 represents the average length of uninterrupted sequences of capital letters. Feature 56 represents the longest uninterrupted sequence of capital letters. While feature 57 is the number of capital letters in the e-mail. The dataset is divided into two classes, spam (unwanted) and ham (legitimate) emails. In an adversarial contest, the attacker changes the number of words and the length of the uninterrupted sequence of capital letters to make a spam email misclassified as ham and vice versa.
- **Breast Cancer** (full name Breast Cancer Wisconsin (Diagnostic) Data Set)

[19] is a dataset that represents the fundamental features extracted from digitized images of a fine needle aspirate of a breast mass. The dataset is made up of 30 real-valued features. The dataset is binary and the classes are M (malignant) and B (benign). In a context of evasion attack, an adversary modifies the attributes of the analysis to alter the diagnosis of the disease. Beyond the somewhat unreal context, the dataset was used to show the ability of FPF to protect the features better than the other models compared and to correctly classify the instances of the dataset.

- **Binary Wine** is a dataset created from the Wine Data Set [19] dataset. Wine Data Set represents the results of a chemical analysis conducted on three wines grown in the same Italian region but derived from three different cultivars. Chemical analysis measures the quantity of 13 different constituents (as alcohol, magnesium, color intensity) found inside the three types of wine. The classification task involves understanding the origin of the wine by comparing the constituents inside it. Since there are three classes and the proposed model has been developed for binary classification, we have binarized the dataset by merging 2 cultivars (1 and 2). In this case an attacker modifies the constituents of the wine coming from a cultivars to make it seem to come from another cultivars and vice versa.
- **MNIST 0/1**, **MNIST 1/7** and **MNIST 5/6**. MNIST database [35] (Modified National Institute of Standards and Technology database) is a dataset of $70,000$ handwritten digits images in grayscale. Each image has a size of 64×64 pixels for a total of 784 features. Each feature has an integer value ranging from 0 to 255 which identifies the intensity of the pixel. In a context of evasion attacks, an attacker modifies the pixels of the image to transform one digit into another. The datasets MNIST 0/1, MNIST 1/7 and MNIST 5/6 are the binary version of MNIST, in which the model is trained to discriminate only two digits.
  Due to the large number of features and instances, it is infeasible to run the brute-force algorithm with attacker A_k on this dataset. Thus we decided to use these datasets only for fast lower-bound experiments 5.4.

5.1.2 Models

In this thesis we decided to compare FPF with the ensemble learning algorithms RF and RSM (applied to decision trees). Below we describe the training parameters of interest for each model.

- **FPF:** The training parameters of FPF are b, r and l. The parameter b indicates the number of features that the model protects against an attack performed by A_k. The parameter r indicates the number of rounds performed by the algorithm during training. Finally parameter l indicates the maximum number of leaves for each tree inside $\mathcal{T}$. If $l = \infty$, then the number of leaves is chosen by the learning algorithm.

- **RSM-DT:** Random Subspace Methods [5, 29, 6] described in section 3.3 was used to train a forest of decision trees, robust against evasion attacks generated by the attacker A_k. The ensemble resulting from the application of RSM on DT was called RSM-DT. RSM is characterized by two training parameter. First, $p \in [0, 1]$ which specifies the size of the subset of features $\mathcal{F}$ for each tree and second the size of the ensemble. Furthermore, since we have used decision trees, it is possible to specify the maximum number of leaves l of each tree.

- **Random Forest:** Random Forest [10] described in section 1.3.1 has as important parameters the size of the forest $\mathcal{T}$ and the size of the subset produced by the boostrap sampling. For this last parameter we have chosen the value $\sqrt{|\mathcal{F}|}$. RF has the possibility to specity the maximum number of leaves, but by definition RF grows as it can. So we decided to specify $l = \infty$, the default value of this learning algorithm. Random Forest is known to have some good level of robustness thanks to the ensembling of several decision trees.

5.1.3 Attacker's strength

In experiments involving the use of the brute-force algorithm, we used two attackers, A_1 and A_2. The reason why we limited the attack to a maximum of 2 features is given by the computational cost of the brute-force algorithm in the calculation of all possible attacks. In the experiments related to fast lower-bound the attacker had a budget greater than 2. We were able to increase the budget because this certificate allow us to approximate the robustness of the model efficiently even with high values of r, b and k. Hereinafter for simplicity when a model classifies orignal instances, it is considered as a model that classifies instances perturbed by an attacker A_0 with budget 0. We have collected the accuracy of A_0 because even if in an adversarial scenario the attacker has no reason for non conducting an attack, the model must still guarantee a high accuracy on non-perturbed instances submitted by a legitimate user. In addition, we referred to the attacker's model A_b as A_k to avoid confusion between the parameter b chosen to train the model and the attacker's strength.

5.2 Evaluation

In a context without attacks it is possible to measure the performance of a model through the *accuracy* denoted ad Acc.

$$Acc = \frac{|\mathcal{D}_{test}| - |E|}{|\mathcal{D}_{test}|} \tag{5.1}$$

whit $E = \{(\boldsymbol{x}, y) \in \mathcal{D}_{test} \mid \mathcal{T}(\boldsymbol{x})y < 0\}$. In a context with an opponent, accuracy is no longer a valid metric for measuring model performance (robustness). To consider an instance in $\boldsymbol{x} \in \mathcal{D}_{test}$ correctly classified, the model must also correctly classify every perturbation $\boldsymbol{x}' \in A_k(\mathcal{D}_{test})$. For this reason, to measure the robustness of the model, we decided to use the *accuracy under attack*, defined as follows.

$$Acc_{A_k} = \frac{|\mathcal{D}_{test}| - |E \cup E'|}{|\mathcal{D}_{test}|} \tag{5.2}$$

with $E' = \{(\boldsymbol{x}, y) \in \mathcal{D}_{test} \setminus E \mid \exists \boldsymbol{x}' \in A_k(\boldsymbol{x}),\ y \cdot \mathcal{T}(\boldsymbol{x}') < 0\}$ the set of the perturbed instances that are not classified correctly by $\mathcal{T}$. In our case, to compare the real robustness of the models, we have implemented a **brute-force** algorithm which generates all possible perturbation of $\boldsymbol{x}$ with respect to A_k and this allows to correctly compute Acc_{A_k}.

5.2.1 Evaluation algorithm

To accurately verify the robustness of the models with respect to the attacker A_k we decided to implement a brute-force attack algorithm which generates all the possible perturbations for the model to be tested. Given the instance $\boldsymbol{x}$ and the number k of features to attack, the brute-force algorithm finds all the possible perturbations $\boldsymbol{x}' \in A_k(\boldsymbol{x})$. The instance $\boldsymbol{x}$ evades the model if there is at least one instance $\boldsymbol{x}' \in A_k(\boldsymbol{x})$ such that $y \cdot \mathcal{T}(\boldsymbol{x}') < 0$. Considering that each $x^{(f)}$ can take any value in $\mathbb{R}$, the size of $A_k(\boldsymbol{x})$ is infinite. On the other hand we can limit its enumeration to the set of attacks that are relevant for the given forest $\mathcal{T}$, i.e., those attacks that can invert the outcome of a test in some internal nodes of trees in $\mathcal{T}$. Recall that nodes in a tree are in the form $x^{(f)} \leq v$ for some threshold v. Indeed, the thresholds used in the tree nodes induce a discretization of the input space $\mathcal{X}$ that we exploit as follows. For any given feature $f \in [1, d]$, we define with $\mathcal{V}_f$ the set of all threshold values v of the nodes in the ensemble, that split instances using f as follows:

$$\mathcal{V}_f = \{\exists \sigma(f, v, t_l, t_r) \in t, t \in \mathcal{T}\} \cup \{\infty\} \tag{5.3}$$

The set $\mathcal{V}_f$ includes all the thresholds that are associated to f in any node $\sigma\{f, v, t_l, t_r\}$ of any tree in $\mathcal{T}$, plus the ∞ value that allows to traverse the right branch of the node with the largest threshold. The set of all possible $\mathcal{V}_f$ is defined as $\mathcal{V}$. Given an attacker A_k, the set of relevant perturbations is thus given by the cartesian product of sets $\mathcal{V}_f$ for k different features. Let $\mathcal{F}_k$ be the set of all subsets $\mathcal{F}_k^{(i)} \subseteq \mathcal{F}$ having size at most k and $|\mathcal{F}_k| = C_{(d,k)}$, we denote with $\hat{A}_k(\boldsymbol{x})$ the set of such perturbations, formally defined as:

$$\hat{A}_k(\boldsymbol{x}) = \left\{ \boldsymbol{x}' \mid x^{(f)} = v, \forall v \in \mathcal{V}_f, \forall f \in \mathcal{F}_k^{(i)}, \forall \mathcal{F}_k^{(i)} \in \mathcal{F}_k \right\}. \tag{5.4}$$

We conclude that an attacker A_k can successfully perturb an instance $\boldsymbol{x}$ against a forest $\mathcal{F}$ if there exists at least one $\boldsymbol{x}' \in \hat{A}_k(\boldsymbol{x})$ for which $\mathcal{T}(\boldsymbol{x})y < 0$. This brute-force approach is very expensive, due to three factors: *i)* as k increases, the number of feature combinations $\mathcal{F}_k$ increases; *ii)* as the number of trees grows, the number of threshold values associated with each feature increases; *iii)* for each perturbed instance $\boldsymbol{x}'$, the prediction $\mathcal{T}(\boldsymbol{x}')$ must be retrieved by traversing the given forest. Evaluating the accuracy of a model in presence of an attacker is a difficult and computationally expensive task. This is due to the possibly large size of $A_b(\boldsymbol{x})$ for some $\boldsymbol{x} \in \mathcal{X}$ and to the number of interactions among trees in a forest. In [14] they showed that verifying the robustness of a forest $\mathcal{T}$ with at most l leaves per tree has cost $\min\{O(l^{|\mathcal{T}|}), O((2|\mathcal{T}|l)^{|\mathcal{F}|})\}$ assuming a L_∞-norm attacker. In [32] they proved that in case of a L_0-norm attacker, as in this work, the problem of finding a successful attack is NP-complete. For simplicity we decided to implement our version of a brute-force algorithm instead of using the algorithm proposed in [32] The algorithm 8 shows the pseudocode of the brute-force algorithm we used to generate all the possible perturbations of a given instance $\boldsymbol{x}$. The algorithm returns `True` when there exists a perturbation $\boldsymbol{x}'$ of $\boldsymbol{x}$ such that $y \cdot \mathcal{T}(\boldsymbol{x}') < 0$, `False` otherwise. To reduce the execution time of the algorithm we used two strategies: *i)* early termination, as soon as a perturbation is found such that $y \cdot \mathcal{T}(\boldsymbol{x}') < 0$, the algorithm ends; *ii)* through the `FastLowerBound` certificate we have filtered the instances for which there is no attack of A_k that evades $\mathcal{T}$. In particular, we only performed brute-force on instances that fast lower-bound consider attackable. We could have also used ALB in cascade but since they have almost similar performances we have chosen not to introduce another exponential component in the algorithm. It should be noted that thanks to the use of early termination, the most difficult instances to process are those that do not have a perturbation that evades the model and for which the whole set of possible attacks must be inspected. Fast lower-bound removes a large number of instances whose set $\hat{A}_k(\boldsymbol{x})$ should be completely inspected. Early termination reduces the search in attackable instances, while fast lower-bound reduces the search in non-attackable instances. The combination of the two strategies significantly reduce the execution time of the BruteForce algorithm.

BruteForce complexity To calculate the temporal complexity of BruteForce we need to look at the pseudocode of the algorithm in 8. The algorithm starts by calling the `FastLowerBound` function on the input instance $\boldsymbol{x}$. As we previously specified the complexity of FLB is $O(bhr+dr+d\log(d))$, so the temporal complexity of BruteForce is at least this. The algorithm continues by executing the **for** inside the function `BruteForce`. The **for** iterates for each element of the set $\mathcal{F}_k$, which has size $C_{(d,k)}$. For each iteration of the **for** the set $\hat{\mathcal{V}}$ is created at a cost of $O(k)$ if $\mathcal{V}$ is implemented as a vector of size d. Furthermore, for each iteration the algorithm makes a call to `BruteForceRec`, a recursive function that calculates all perturbations $\boldsymbol{x}'$ of $\boldsymbol{x}$ given all threholds sets in $\hat{\mathcal{V}}$. Function `BruteForceRec` recurs for every $\mathcal{V}_f \in \hat{\mathcal{V}}$, so k times. Each recursion performs a **for** on the thresholds $v \in \mathcal{V}_f$. The size of $\mathcal{V}_f$ is variable for each feature f. Since we are calculating an upper bound of the temporal complexity we define $m = \max\{|\mathcal{V}_f| \mid \forall\, \mathcal{V}_f \in \mathcal{V}\}$, as the size of the largest set of thresholds. Finally for any possible evading instance $\boldsymbol{x}'$ the prediction $\mathcal{T}(\boldsymbol{x})$ is calculated at a cost of $O(r(2b+1)h)$. The final temporal complexity of BruteForce in the worst case is $O(bhr + dr + d\log(d) + C_{(d,k)}(k + m^k \cdot rh(2b+1)))$.

5.3 Parameters analysis

Each model used in the experiments has different training parameters. Before comparing the robustness of the models against A_0, A_1 and A_2, we performed experiments to find the training parameters that generate models with the highest robustness. In this way the comparison between the state of the art and the FPF model was fair. For RF and RSM-DT we investigated the impact of the number of trees in the ensemble with the robustness of the model. For FPF and RSM-DT we analyzed the importance of the maximum number of leaves per tree. For FPF we looked for the value of p which generates the most robust model. Finally, for the FPF model we investigated the effects of the training parameters b and r. Due to the complexity of the experiments (which involve the calculation of the accuracy under attack with the brute-force algorithm), we decided to conduct the analysis of the training parameters only on the Breast_Cancer dataset. We chose this dataset among those listed above because it has a good trade-off between number of features/instances, small enough to perform all the experiments of the analysis and large enough to guarantee relevant results for each parameter. As the parameters were estimated, we performed the remaining experiments specifying these parameters with the best estimated value.

Algorithm 8 BruteForce

```
Require: The instance to be verified $\boldsymbol{x} \in \mathbb{R}^d$, the corresponding label $y$, the
    attacker budget $k$, tree ensemble $\mathcal{T}$, the set $\mathcal{V}$ and the set $\mathcal{F}_k$.
 1: function BruteForce($\boldsymbol{x}$,$y$,$k$,$\mathcal{T}$,$\mathcal{F}_k$)
 2:     if FastLowerBound($\boldsymbol{x}, y, k$) $\leq 0$ then
 3:         for $\mathcal{F}_k^{(i)} \in \mathcal{F}_k$ do
 4:             $\hat{\mathcal{V}} = \{\mathcal{V}_f | \mathcal{V}_f \in \mathcal{V} \wedge f \in \mathcal{F}_k^{(i)}\}$
 5:             if BruteForceRec($\boldsymbol{x}, y, \mathcal{T}, \hat{\mathcal{V}}$) then
 6:                 return True
 7:             end if
 8:         end for
 9:     end if
10:     return False
11: end function

12: function BruteForceRec($\boldsymbol{x}'$,$y$,$\mathcal{T}$,$\hat{\mathcal{V}}$)
13:     if $\mathcal{V}' = \emptyset$ then
14:         return $\mathcal{T}(\boldsymbol{x}') \neq y$
15:     else
16:         for all $v \in \mathcal{V}_f, \mathcal{V}_f \in \hat{\mathcal{V}}$ do
17:             $\boldsymbol{x}'^{(f)} = v$
18:             if BruteForceRec($\boldsymbol{x}', y, \mathcal{T}, \hat{\mathcal{V}} \setminus \{\mathcal{V}_f\}$) then
19:                 return True
20:             end if
21:         end for
22:         return False
23:     end if
24: end function
```

5.3.1 Features subset size analysis

As previously mentioned, RSM-DT is characterized by the parameter p which specifies the size of the subset of $\mathcal{F}$ inside each base-learner. The p parameter is critical to the accuracy and robustness of the ensemble. A small value of p creates a base-learner with few features that cannot generalize the distribution of data and therefore has an underfitting problem. On the other hand, if the value of p is too large, there will be many more features inside and in common between base-learners. This allows the adversary to have multiple attack opportunities and to attack multiple base-learners simultaneously with the modification of the same feature. Thus the p value is fundamental for the trade-off between accuracy and robustness. In this regard, we analyzed the impact of parameter p on robustness of the model in the following way. To conduct the analysis we trained a RSM-DT for each value of $p \in \{0.1, 0.2, 0.4, 0.6\}$ and for each of the following parameters: $l = \infty$, $|\mathcal{T}| = \{10, 100, 300, 500\}$. For each model we calculated the accuracy under attack with respect to attackers A_k with $k \in \{0, 1, 2\}$. It is important to note that when p is small enough the resulting ensemble can present a distribution of features within the ensemble similar to FPF. In fact, as explained in the Bagging section 1.3.1, using boostrap sampling to sample a subset of features with a small size allows the ensemble to train trees with few features in common. This fact is similar to the distribution achieved with robust partitioning in FPF. In addition, if we consider that our model is trained for several rounds, the distribution of the features can be very similar. This similar distribution leads to similar performances between the two models. however RSM-DT doesn't have robustness property by construction and this can be seen in the results with lower performances than FPF.

5.3.2 Maximum number of leaves analysis

For a decision tree the maximum number of leaves is a fundamental parameter both in a normal context and under attack. If the tree has few features to discriminate the samples, limiting the number of leaves prevents the model from overfitting by growing too much. Furthermore, a limited number of leaves generate a reduced number of iternal nodes and therefore less opportunity for the adersary to attack the model. For each model we made some considerations on which is the best number of leaves to choose. We decided to train Random Forest with a maximum number of leaves $l = \infty$, which corresponds to the standard implementation. In general RF grow each tree as high as necessary to maximize forest accuracy on legitimate instances. In addition, each tree in the forest sees all the features in $\mathcal{F}$, performing boostrap sampling for each node. If we limited the number of leaves in RF, each tree would be prevented from exploiting all the features. In FPF and RSM, each tree uses a subset of features, however, each algorithm use its strategy for the distribution of the features and choosing the size of the subset. The experiments on the leaves have allowed us to understand for the two models which is the most suitable value of the parameter l. For both models, we analyzed the accuracy under attack for each $l \in \{\infty, 4, 8, 16\}$. For FPF we analyzed the effect of the number of leaves on the robustness for each $b \in \{1, 2, 3, 4, 5\}$ and $r = 30$. We decided to vary b to analyze the influence of the l parameter on decreasing features in each tree. For RSM-DT we analyzed the influence of the number of leaves as the size of the ensemble increases, $|\mathcal{T}| = \{100, 300, 500\}$, to see if the choice of the best number of leaves changes as the forest grows.

5.3.3 Forest size analysis

As mentioned above, the ensemble learning algorithms RF and RSM-DT have the training parameter $|\mathcal{T}|$ which defines the size of the resulting ensemble (i.e. the number of trees inside the forest). In Random Forest it is well known that increasing the size of the forest increases the accuracy of the model in a context without attacks. After reaching a certain number of trees, the forest's performance remains stable. For RSM it has been shown that increasing the size of the forest allows to have a better accuracy both in a context without attacks [29] and under attack [5]. In general it is reasonable to think that as the number of trees increases, the robustness of the model increases. This is because the attacker has more difficulty finding an attack that evades the majority of trees. To empirically demonstrate that forest robustness increases with the number of trees, we tested the accuracy under attack of RF and RSM-DT as the size of the ensemble increased. The attacks were generated by A_k with $k \in \{0, 1, 2\}$ and the robustness of the models was analyzed for each $|\mathcal{T}| \in \{1, 3, 5, 10, 30, 50, 100, 300, 500\}$. The models were trained with the following training parameters: $l = \infty$, and $p = 0.2$ for RSM-DT.

5.3.4 Rounds analysis

As for RF and RSM-DT we expect that as the number of trees inside FPF increases, the accuracy of the model will also increase both under attack and not. Feature Partitioned Forest does not have the training parameter $|\mathcal{T}|$, but since the number of trees inside it is $r(2b+1)$ the size of the ensemble depends on b and especially on r. So to demonstrate that the increase in the number of trees within the ensemble improves forest performance it is necessary to separate the analysis of r from b. The b parameter specifies how many features to protect and is not to be considered as a parameter to enlarge the forest. Furthermore, the results obtained by analyzing b as a parameter to make the forest larger can lead to misleading results. In fact, with the increase of b the size of the forest increases, but the features inside the trees decrease and with this also the accuracy of the base-learner, with consequent lower performances. The r parameter was instead designed to increase the gap between attackable and non-attackable trees, to distribute the features between the trees in the ensemble in different ways and to increase the number of trees to be broken by attacking the same number of features. However, it is still important to show that increasing r increases the robustness of the model for each b. For this reason we have trained a model for each $r \in \{1, 15, 30\}$ and for each $b \in \{1, 2, 3, 4, 5\}$, with $l = 8$. The robustness was calculated based on the accuracy under attack calculated with respect to the attackers A_k with $k \in \{0, 1, 2\}$. One last important consideration to make to understand the results of these experiments. Models are trained to protect b features with b ranging from 1 to 5 but are attacked at most on two features. Because of this the effect of the growth of r is expected to be more on models trained to be robust to attacks on 1 or 2 features. For models trained to protect 3, 4 and 5 features, the effect of r is expected to be less prominent as the model is trained to be robust to stronger attacks. However, a slight improvement in robustness can still be seen as r increases. In particular, we expect to see a more noticeable effect of the r passing from $r = 1$ to $r = 15$ for the reason that we have explained in section 4.2.2: as r increases, the algorithm introduces $\lceil \frac{d}{2} \rceil - 1$ trees gap between possible attackable and non-attackable trees.

5.3.5 Model robustness parameter analysis

As explained in Feature Partitioned Forest chapter, the parameter b is used to specify how many features the model must protect during training. The analysis on parameter b that have been defined in this section allow to empirically demonstrate the following considerations:

i) since the trees inside the forest are not perfectly accurate, it is possible that the robustness by construction is invalidated by the presence of errors independent of the attacks. Given an attacker A_k, it is reasonable to think that training a model m_1 with $b_1 > k$ could lead to better performances than m_2 trained with $b_2 \geq k$, with $b_1 > b_2$. This because m_1 protects more features than m_2 and therefore has more trees that cannot be attacked by A_k.

ii) As the parameter b grows, the number of features in each tree decreases and consequently its accuracy. For this reason, as the b increases, the model is expected to have a decrease in accuracy on original instances.

iii) By increasing the number of trees in the forest it is possible to solve the problems of the first and second observations. As the number of trees increases, the probability that independent errors are the cause of the misclassification decreases. So models trained for b smaller will increase their robustness and since they have more features per tree they will maintain a high accuracy on original instances. This last observation has similarities with what was said in the round analysis section, but this time is analyzed from the point of view of the parameter b. To demostrate the first observation we trained a model of about 50 trees for each $b \in \{1, 2, 3, 4, 5\}$ and $l = 8$ and we calculated the accuracy under attack with A_k with $k \in \{0, 1, 2\}$. To demostrate the third observation we also repeated the experiments with 100 and 300 trees to show how models trained with smaller b acquire greater robustness for less strong attacks (A_1). The second observation can be appreciated in all experiments against A_0 with the increase of b. In this experiment we chose the rounds for each b to get more or less 10, 50, 100 and 300 trees per model. To have the desired number of trees we calculated $r = \lfloor \frac{|\mathcal{T}|}{2b+1} \rfloor$. The reasons why we have chosen to keep a constant number of trees for each model trained on a different b are two: The first is to focus the analysis only on the parameter b, without the influence of r. The second one, for the same number of trees, a model with large b has less attackable trees. For example, given two models, one trained with $b = 1$ and $r = 5$ and one with $b = 2$ and $r = 3$, both have 15 trees, but against A_1 the first has at most 5 attackable tree, the second at most 3. This is a property of parameter b. But if we had specified $r = 5$ also for the second model, the number of possible attackable trees would have been 5 out of 25, an even bigger and more unfair difference of trees that cannot be attacked compared to the first model.

5.4 Certificates analysis

To show the pecision and usefulness of the lower-bounds, we have prepared two types of analysis. The first shows the precision of the two lower-bound algorithms (accurate and fast lower-bound) with respect to the real robustness of the model. While the second analysis shows how it is possible to calculate in advance a lower-bound of the accuracy under attack for a model that would be infeasible to verify with the brute-force algorithm due to the high number of combinations $C_{(d,b)}$ and number of thresholds v for each f.

Certificates accuracy In FPF chapter we presented two certificates of robustness, called fast lower-bound and accurate lower-bound. What we wanted to show with these experiments is how the two certificates allow to accurately approximate the accuracy under attack of the FPF model without attacking it with the brute-force algorithm. In isolated contexts we referred to the accuracy approximated by the two lower-bounds as accuracy lower-bound, in the cases of comparison with BruteForce instead we called it accuracy under attack for simplicity. The analysis was conducted on the Breast_Cancer, Binary_Wine and Spam_Base datasets, varying b. For the first two datasets we trained a model with about 300 trees. Due to the large number of features and instances in Spam_Base, the complexity for calculating the Acc_{A_k} with BruteForce is very high and therefore we decided to perform the analysis with about 100 trees. For each model we calculated Acc_{A_k} with both brute-force and lower-bounds strategies, with $k \in \{1, 2\}$.

Fast lower-bound utility Thanks to the speed of the fast lower-bound algorithm we have been able to show the trend of the lower bound accuracy of the models generated with FPF also for attacker A_k with budget $k > 2$. The purpose of the experiment is to show the behavior of the FPF algorithm with the variation of b and k and show the potential of the fast lower-bound algorithm by calculating the accuracy lower bound with a large number of b and k and even with a large number of instances and trees inside the forest. Below we have reported the values of the training parameters used to conduct the experiments.

- Breast_Cancer: $\mathcal{T} \approx 1000$, $l = 8$, $b \in \{1, \dots, 14\}$ and A_k with $k \in \{0, \dots, 14\}$.
- Spam_Base: $\mathcal{T} \approx 1000$, $l = 8$, $b \in \{1, \dots, 14\}$ and A_k with $k \in \{0, \dots, 14\}$.
- Binary_Wine: $\mathcal{T} \approx 3000$,$l = 8$, $b \in \{1, \dots, 6\}$ and A_k with $k \in \{0, \dots, 6\}$.
- MNIST 0/1, MNIST 1/7 and MNIST 5/6: $\mathcal{T} \approx 500$, $l = 8$, $b \in \{1, \dots, 100\}$ and A_k with $k \in \{0, \dots, 100\}$.

Furthermore, the experiment wants to show that as the training parameter b increases, the accuracy of the model on the original instances and on weaker attacks decreases compared to models trained for less strong attacks. This is because as b increases, the number of features per tree decreases, with consequent loss of accuracy. At the same time, the experiment wants to show the greater robustness at strong attacks in models trained with large b.

5.5 Models comparison settings

As reported in the next chapter of the results of the experiments, in general all the models perform better by increasing the size of the forest. The problem with training a model with many trees is calculating the accuracy under attack with BruteForce. As $\mathcal{T}$ increases the number of thresholds v associated with each feature f grows and calculates the set of all possible attacks $\mathcal{Z}$ through the BF algorithm becomes infeasible. For this reason, for our algorithm FPF we decided to train the models keeping a fixed number of trees for every b. Specifically, we chose a total of $|\mathcal{T}| = 300$ for Breast_Cancer and Binary_Wine and $|\mathcal{T}| = 100$ for Spam_Base. For RF and RSM-DT we trained a model at the maximum number of trees that we could test and one with the same number of trees used for FPF to compare the models with the same forest size. Precisely for each algorithm, RF and RSM-DT, we have chosen the following forest size. For Breast_Cancer we trained a model with 500 and 300 trees, and for Binary_Wine with 1000 and 300 trees. Finally for Spam_Base we trained a model of 100 trees. The remaining training parameters used for each model are shown below. For FPF we trained a model to protect $b \in \{1, 2, 3, 4, 5\}$ features, with a maximum number of leaves $l = 8$. For RSM-DT we have specified $p = 0.2$ as the proportion of the subset of $\mathcal{F}$ used in each tree and with $l = \infty$ the maximum number of leaves. Finally for RF we have specified $l = \infty$ and $\sqrt{|\mathcal{F}|}$ as the size of the of the boostrap sampling for each node.

Summary

In this chapter we have defined the experimental setting to verify the robustness of FPF, RF and RSM-DT with respect to the attacks generated by A_k with budget $k \in \{0, 1, 2\}$. Furthermore, we have defined the experiments to show the performance and usefulness of the ceritifcates. Briefly, the chapter can be summarized as follows:

- **Experimental settings.** In this section we have presented the datasets used in the experiments (**Breast_Cancer**, **Binary_Wine**, **Spam_Base**, **MNIST 0/1**, **MNIST 1/7** and **MNIST 5/6**) and for each we have given: a general description, the number of instances/features, the majority class percentage, the portion of training and test set, etc.). Then we defined the models on which we have performed the experiments: **FPF**, **RF** and **RSM**. Finally, we defined the attacker's strength (budget) we have used to evaluate the robustness of the models and the precision of the certificates.

- **Evaluation.** In the evaluation section we introduced the **accuracy under attack**, the metric to evaluate the robustness of the model. Accuracy under attack represents the ratio of instances in a given set that the model correctly classifies under attack. Finally we defined the **brute-force** algorithm denoted ad BruteForce, used to generate all possible attacks and verify the robustness of the model. Together with the definition of the BruteForce algorithm, we have given its temporal complexity.

- **Parameters analysis.** The section called parameters analysis defines all the experiments performed for tuning the models and to investigate their behavior when the parameters change. For each algorithm we analyzed in detail the robustness of the models with respect to their training parameters ($|\mathcal{T}|$, l, b, r, p).

- **Certificates analysis.** In this section we have defined the experiments performed to prove empirically the precision of the lower-bounds despite their approximation and some experiments aimed at showing the behavior of FPF compared to an attacker with budget $k > 2$, which would be infeasible to be calculated without lower-bound.

- **Models comparison settings.** In the last section of the chapter we defined the values of the training parameters of each model used in the comparisons. Precisely for FPF: $l = 8$ and $b \in \{1, 2, 3, 4, 5\}$. For RSM-DT: $p = 0.2$ and $l = \infty$. For RF: $l = \infty$ and boostrap samplig size $\sqrt{|\mathcal{F}|}$. The forest size was chosen based on the maximum size that BF can feasibly test for each dataset.

Chapter 6

Experimental results

In this chapter we reported and described the results of the experiments defined in the previous chapter. The chapter is divided into three macro parts. In the first part, for each model, we described the results of the experiments conducted for tuning the training parameters. In the second part, we compared the robustness of the FPF, RSM-DT and RF on Breast_Cancer, Binary_Wine and Spam_Base datasets, trained with the best parameters found in the previous experiments. Finally, we described the results of the analysis conducted on fast and accurate lower-bounds.

6.1 Parameters analysis

In this first part of the chapter we have collected all the results of the experiments on the training parameters performed on the dataset Breast_Cancer. For each model, FPF, RSM-DT and RF, we have summarized the results below.

6.1.1 FPF analysis

For our proposed algorithm, Feature Partitioned Forest, we analyzed the maximum number of leaves l, the number of features to be protected b, and the number of rounds r. Below, for each parameter we report the results of the analysis.

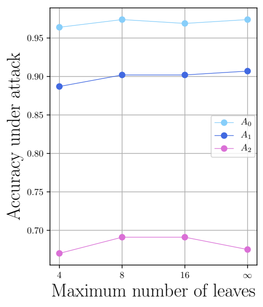
Maximum number of leaves analysis

The first analysis we performed was on the maximum number of leaves for the decision trees of the forest. Recalling what was said in the evaluation methodology section, fixed $r = 30$, we analyzed the behavior of the model attacked by A_k with budget $k \in \{0, 1, 2\}$, by varying $b \in \{1, 2, 3, 4, 5\}$ for each maximum number of leaves $l \in \{4, 8, 16, \infty\}$. Table 6.1 summarizes the accuracy under attack collected

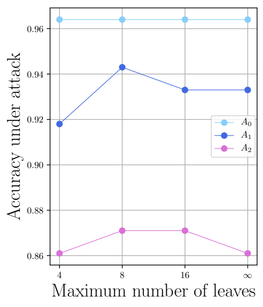
Table 6.1: Analysis on the maximum number of leaves l: Breast_Cancer dataset, FPF model, number of rounds $r = 30$. For each value of b and A_k the highest accuracy under attack associated with the maximum number of leaves l used to train the model is highlighted in **boldface**.

b	A_k	l			
		4	8	16	∞
1	0	0.964	**0.974**	0.969	**0.974**
	1	0.887	0.902	0.902	**0.907**
	2	0.670	**0.691**	**0.691**	0.675
2	0	**0.964**	**0.964**	**0.964**	**0.964**
	1	0.918	**0.943**	0.933	0.933
	2	0.861	**0.871**	**0.871**	0.861
3	0	**0.964**	0.954	0.954	**0.964**
	1	0.912	**0.938**	0.923	0.923
	2	0.887	**0.902**	0.892	0.881
4	0	**0.964**	0.954	0.954	0.954
	1	0.918	**0.933**	0.923	0.929
	2	0.892	**0.902**	0.897	0.887
5	0	**0.959**	0.948	0.954	0.954
	1	0.912	**0.928**	0.917	0.917
	2	0.892	**0.897**	0.892	0.887

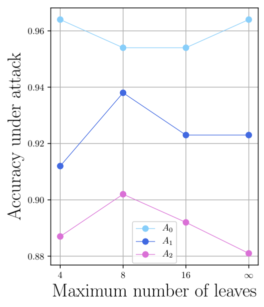
for each model, instead in Figure 6.1 it is possible to see a graphical representation of the results for each b. As can be seen from the results, with $b = 1$ the robustness of FPF is not affected much by the parameter l. But as b increases, $l = 8$ turns out to be the limit of the number of leaves that produces models with greater robustness under attack, both for A_1 and for A_2. For A_0 and low values of b, the best value of l appears to be ∞. This was predictable beacause each tree inside the forest can choose the best number of leaves to discriminate training data (legitimate instances), so that each tree is grown to best discriminate non-attacked instances. However, for large values of b the model risks overfitting on training data if the leaves are not limited. In fact for $b = 4$ and $b = 5$, with $l = 4$ the performances are better than with $l = \infty$. Obviously a high accuracy of the model on original instances does not give any guarantee of robustness under attack. Thus, with an adequately limited number of leaves, there is a decrease in accuracy on original instances but the robustness of the model under attack increases. For the subsequent experiments on FPF we chose $l = 8$ as the best maximum number of leaves.



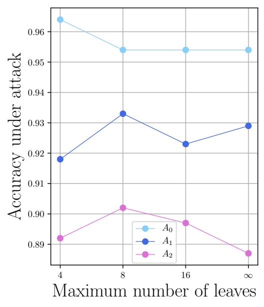
(a) FPF $b = 1$



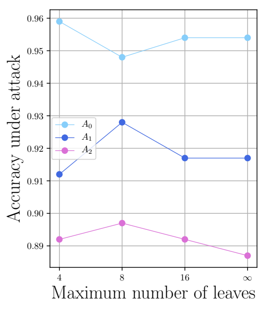
(b) FPF $b = 2$



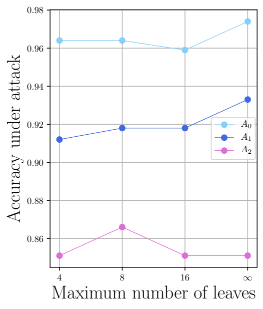
(c) FPF $b = 3$



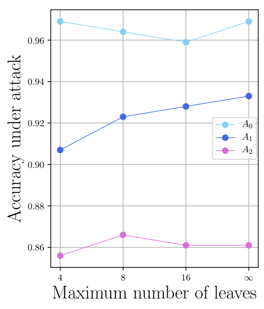
(d) FPF $b = 4$



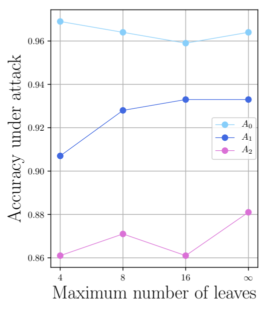
(e) FPF $b = 5$



(f) RSM-DT $|\mathcal{T}| = 100$



(g) RSM-DT $|\mathcal{T}| = 300$



(h) RSM-DT $|\mathcal{T}| = 500$

Figure 6.1: FPF and RSM-DT leaves analysis on Breast_Cancer dataset.

Rounds analysis

In this section we reported the results of the analysis on the parameter r which specifies the total number of trees, i.e. how many sub-forests of size $2b + 1$ are inside $\mathcal{T}$. Table 6.2 contains the results of the experiments performed on varyng r. Fixed $l = 8$ we trained FPF models for each $r = \{1, 15, 30\}$ and $b = \{1, 2, 3, 4, 5\}$. As expected from the considerations made in the chapter of the experiments, by increasing the number of rounds the robustness of the forest increases. A clear example of the effect of r can be seen when a model is trained with $b = 1$. The model trained with $r = 1$ has a forest of size 3. By attacking the ensemble on 2 features, the model gets $Acc_{A_2} = 0.108$, evidence that the attacker can in most cases break 2 trees out of 3. By increasing the round value to $r = 15$, the size of the forest becomes 45 trees. In this case $Acc_{A_2} = 0.670$, a high value considering that the model has been trained to be robust to attacks on a single feature. The increase in accuracy under attack from $r = 1$ to $r = 15$ is 83.9%. While from $r = 15$ to $r = 30$ with the same attacker it is 3%, a slight increase, a sign that the model has stabilized. With a higher b, for example $b = 3$ going from $r = 1$ to $r = 15$ with A_2 there is an increase of 5.4% while from $r = 15$ to $r = 30$ with the same attacker

Table 6.2: Analysis on the number of rounds r: Breast_Cancer dataset, FPF model, $l = 8$. For each value of b and A_k the highest accuracy under attack associated with the number of rounds r used to train the model is highlighted in **boldface**.

b	A_k	r		
		1	15	30
1	0	0.964	0.969	**0.974**
	1	0.881	**0.902**	**0.902**
	2	0.108	0.670	**0.691**
2	0	0.954	**0.964**	**0.964**
	1	0.902	**0.943**	**0.943**
	2	0.804	0.866	**0.871**
3	0	**0.954**	**0.954**	**0.954**
	1	0.923	**0.938**	**0.938**
	2	0.851	0.897	**0.902**
4	0	0.943	**0.954**	**0.954**
	1	0.907	0.928	**0.933**
	2	0.871	0.887	**0.902**
5	0	0.933	**0.954**	0.948
	1	0.912	0.923	**0.928**
	2	0.876	0.892	**0.897**

is 0.5%. These results confirm what was said in the section of experiments. As the value of r increases, the robustness of the model also increases. Beyond a certain value of r, the robustness of the model stabilizes. Given the attackers A_1 and A_2 as b increases the influence of r decreases. This phenomenon occurs because the model is trained to be robust to stronger attacks, so even with small r the model is robust. In conclusion, as expected accuracy under attack increases when increasing the number of rounds until a plateau is reached. So training a model using a large number of rounds r improves its robustness.

Model robustness parameter analysis

The last analysis conducted on FPF is related to the parameter b. We have trained a model for each each value of $b = \{1, 2, 3, 4, 5\}$ and ensemble size $|\mathcal{T}| = \{10, 50, 100, 300\}$. The impact of b is summarized in Table 6.3. These experiments show the behavior of FPF by varying b with a constant number of trees. In this case a trade-off is apparent. Even if increasing b is expected to increase robustness against stronger attacks, at the same time it reduces the number of features that can be exploited when training a single tree. This harms the performance of the whole ensemble especially with larger values of b. This would be more apparent with datasets that contain a limited number of informative features. The results

Table 6.3: Analysis on training parameter b: Breast_Cancer dataset, FPF model, $l = 8$. For each value of $|\mathcal{T}|$ and A_k the highest accuracy under attack associated with the value of b used to train the model is highlighted in **boldface**.

$\lvert\mathcal{T}\rvert$	A_k	b				
		1	2	3	4	5
10	0	**0.964**	0.943	0.954	0.943	0.933
	1	0.871	0.897	0.912	**0.923**	0.912
	2	0.340	0.840	0.851	0.871	**0.876**
50	0	**0.969**	**0.969**	0.964	0.954	0.948
	1	0.902	0.938	0.938	**0.943**	0.928
	2	0.628	0.871	0.876	**0.892**	**0.892**
100	0	**0.974**	0.964	0.954	0.954	0.948
	1	0.912	**0.943**	0.938	0.933	0.928
	2	0.675	0.871	**0.902**	0.897	0.897
300	0	**0.974**	0.964	0.954	0.954	0.948
	1	0.912	**0.943**	0.938	0.933	0.928
	2	0.711	0.876	0.897	**0.902**	0.897

show that when using a limited number of trees, accuracy under attack increases with b. For example with $|\mathcal{T}| = 10$ and A_1 the robustness of the model trained with $b = 5$ improves by 0.041 compared to the model trained with $b = 1$. Against A_2 the robustness increases by 0.036 by varying b from 2 to 5 and by 0.536 from 1 to 5. The increase in robustness is due to what we said in section 5.3.5. If we train two models, respectively with $b = b_1 \in \mathbb{N}^+$ and $b = b_2 \in \mathbb{N}^+$ with $b_1 < b_2$. For the same number of trees, the model trained with b_1 has more attackable trees than the model trained with b_2. For this reason, in the first model an independent error is more likely to change the result of the forest prediction. But when the forest is sufficiently large to exploit the ensembling benefits, then the limited accuracy of singles trees plays an important role making not rewarding the use of larger values of b. For instance, in Breast_Cancer dataset, we have identified only 15 informative features which are difficult to partition in $2b + 1$ sets for $b = 5$. In support of this with $|\mathcal{T}| = 100$ the model trained for $b = 2$ has an $Acc_{A_1} = 0.943$ instead the model with $b = 5$ has an $Acc_{A_1} = 0.928$. We conclude that, while it is beneficial to increase the number of rounds r, it is not always a good strategy to increase b, unless the dataset we have at hand has a sufficiently large set of informative features compared to the attacker strength.

6.1.2 RSM-DT analysis

In this section we have exposed the results of the experiments conducted to find the best values for the training parameters of RSM-DT. The parameters analyzed are the subset size parameter p, the maximum number of leaves l and the forest size $\mathcal{T}$. Below for each parameter we report the results of the analysis.

Features subset size analysis

The first parameter analyzed is p which controls the size the subset of $\mathcal{F}$ sampled by the boostrap sampling. Each tree inside the forest is trained on a sample of features of size $|\mathcal{F}| \cdot p$. The search for the best subset size was done for every $p \in \{0.1, 0.2, 0.4, 0.6\}$ for every $|\mathcal{T}| \in \{10, 100, 300, 500\}$ and with $l = \infty$ compared to A_k with budget $k \in \{0, 1, 2\}$. In Table 6.4 we have reported the Acc_{A_k} calculated for each model and budget of the attacker. The results of these experiments are particularly explicit. It is clear that RSM-DT acquires greater robustness against A_k with a value of $p = 0.2$. In general it can be seen that with the increase of p the robustness decreases, especially for A_2. This because as p grows, more trees have features in common, even more than one feature, so the adversary involves multiple trees with the same attack. The difference between $p = 0.1$ and $p = 0.2$ is not as evident as for the other values, but as can be seen from the table 6.4, models trained with $p = 0.1$ have lower accuracy on non-attacked istances. Values of $p < 0.2$ generates too small subset that brings trees with few features and poor ability to generalize unknown instances at prediciton time. For all subsequent experiments we kept the value of $p = 0.2$.

Maximum number of leaves analysis

As we did for FPF also for RSM-DT we analyzed the accuracy under attack by varying the number of leaves. For each $l \in \{4, 8, 16, \infty\}$ and for each forest size $|\mathcal{T}| \in \{100, 300, 500\}$ we have trained a model with RSM-DT and calculated the Acc_{A_k} with budget $k \in \{0, 1, 2\}$. The results of this analysis are summarized in Table 6.5 and in Figure 6.1. From the table and the figures it can be seen that as the size of the forest grows it becomes more and more evident that the value of l which produces the highest robustness is ∞ reaching an $A_{A_2} = 0.881$. From the results of this experiment we decided to train all RSM-DT models with maximum number of leaves $l = \infty$.

Forest size analysis

The latest analysis conducted on RSM-DT is on the influence of the number of trees inside the forest. We collected the accuracy under attack of the models generated

Table 6.4: Analysis on the parameter p: Breast_Cancer dataset, RSM-DT model, $l = \infty$. For each value of $|\mathcal{T}|$ and A_k the highest accuracy under attack associated with the subset size p used to train the model is highlighted in **boldface**.

$\lvert\mathcal{T}\rvert$	A_k	p			
		0.1	0.2	0.4	0.6
10	0	0.928	**0.974**	0.948	0.969
	1	0.814	0.845	**0.856**	0.443
	2	0.649	**0.655**	0.216	0.088
100	0	0.943	**0.974**	0.959	0.954
	1	0.912	**0.933**	0.861	0.763
	2	**0.851**	**0.851**	0.325	0.129
300	0	0.954	**0.969**	**0.969**	0.948
	1	0.923	**0.933**	0.892	0.340
	2	0.856	**0.861**	0.309	0.196
500	0	0.954	0.964	**0.974**	0.954
	1	0.923	**0.933**	0.892	0.34
	2	0.861	**0.881**	0.299	0.191

Table 6.5: Analysis on the maximum number of leaves l: Breast_Cancer dataset, RSM-DT model, $p = 0.2$. For each value of $|\mathcal{T}|$ and A_k the highest accuracy under attack associated with the maximum number of leaves l used to train the model is highlighted in **boldface**.

$\lvert\mathcal{T}\rvert$	A_k	l			
		4	8	16	∞
100	0	0.964	0.964	0.959	**0.974**
	1	0.912	0.918	0.918	**0.933**
	2	0.851	**0.866**	0.851	0.851
300	0	**0.969**	0.964	0.959	**0.969**
	1	0.907	0.923	0.928	**0.933**
	2	0.856	**0.866**	0.861	0.861
500	0	**0.969**	0.964	0.959	0.964
	1	0.907	0.928	**0.933**	**0.933**
	2	0.861	0.871	0.861	**0.881**

by RSM-DT with $p = 0.2$ and $l = \infty$, as the ensemble size grows. The analysis results are shown in the Table 6.6. From the results we see that the growth of the forest also increases the robustness against A_1 and A_2. In conclusion, from these results we can confirm that also RSM-DT becomes more robust as the forest grows.

Table 6.6: Analysis on the number of trees $|\mathcal{T}|$: Breast_Cancer dataset, RSM-DT model, $l = \infty$, $p = 0.2$. For each value of $|\mathcal{T}|$ the highest accuracy under attack with respect to the attacker A_k used to test the model is highlighted in **boldface**.

$\|\mathcal{T}\|$	A_k		
	0	1	2
1	0.907	0.0	0.0
3	0.943	0.273	0.005
5	0.948	0.32	0.010
10	0.969	0.845	0.665
30	0.964	0.923	0.809
50	**0.974**	0.912	0.799
100	**0.974**	**0.933**	0.851
300	0.969	**0.933**	0.861
500	0.964	**0.933**	**0.881**

6.1.3 RF analysis

The last model we analyzed is RF. As mentioned previously for this model we have not specified the maximum number of leaves, so by default for the RF learning algorithm it is $l = \infty$. We chose $\sqrt{|\mathcal{F}|}$ for the size of the subset of features sampled by boostrap samplig at each node. For RF we analyzed the influence of forest size on the robustness of the model. We performed the same experiments conducted for RSM-DT. The results of the analysis are reported in the Table 6.7. From the results it can be seen that RF also has an increase in robustness as the forest grows. The larger the forest, the less likely it is to be fooled by attacking one or two features.

6.2 Algorithms comparison

In Tables 6.8, 6.9 and 6.10 we reported the accuracy under attack of FPF, RF and RSM-DT trained as specified in section 5.5, against an attacker that can modify 0, 1 or 2 features. Feature Partitioned Forest provided the highest robustness in all attack scenarios, both with less or equal number of trees than the other algorithms. In all datasets, RF performs best in absence of attacks, but as the attacker budget increases its accuracy under attacks drops significantly. Analyzing each model with respect the accuracy under attack achieved by FPF. For example, for A_1 on the Spam_Base dataset, the FPF model achieves an accuracy under attack of 0.834, while RF gets 0.580, the difference in performance between the two models is 30.4%. Subsequent comparisons are based on this measure. When attacking two features RF has a 76.5% and 81.3% percent decrease in accuracy, rispectively

Table 6.7: Analysis on the number of trees $|\mathcal{T}|$: Breast_Cancer dataset, RF model, $l = \infty$. For each value of $|\mathcal{T}|$ the highest accuracy under attack with respect to the attacker A_k used to test the model is highlighted in **boldface**.

$\|\mathcal{T}\|$	A_k		
	0	1	2
1	0.902	0.170	0.0
3	0.938	0.742	0.0
5	0.948	0.794	0.211
10	0.959	0.840	0.572
30	0.969	0.907	0.763
50	0.969	0.907	0.799
100	0.964	0.907	0.804
300	**0.974**	0.907	0.825
500	0.969	**0.918**	**0.83**

for Spam_Base and Binary_Wine datasets. By construction each tree in a RF can potentially contain all the features, so each attack can involve the whole forest. As the number of attackable features increases, the probability of failure of each tree increases and the strength of RF decreases significantly. This confirms that the L_0-norm attack we are tackling in this work is indeed very powerful and sufficient to fool a very accurate and effective random forest model. The RSM-DT provides good performance in absence of attacks, meaning that the dataset projection is not disadvantageous, and it is much more robust than RF in present of attacks. The performance of RSM-DT is similar to that of FPF when only one feature is attacked, but when two features are attacked FPF shows significantly better performance than RSM-DT. In fact, compared to our FPF algorithm, RSM-DT decreases in accuracy under attack by 12.6% and 10.5% respectively for Spam_Base and Binary_Wine datasets. For Breast_Cancer dataset all three algorithms have very good performances, however FPF is always the best under attack. In Figure 6.2 it is possible to see a graphic representation of the comparison between the models for each of the three datasets. For each algorithm, we took the highest accuracy under attack for each budget. This because given k, an adversary A_k has no advantage in attacking with a lower budget. Therefore it is better to choose the most robust model with respect to the budget k. On the contrary for A_0 we cannot take the maximum value of Acc_{A_0} among all the results in Table 6.8. This because the higher Acc_{A_0} may not correspond to one of the most robust models against A_1 and A_2. A robust model should also be as accurate as possible on original instances. For this reason, for A_0 we took the lower accuracy between the two most robust models against A_1 and A_2 for each learning algorithm. From the figures in 6.2 it is clear that as the budget of A_k increases the algorithm FPF produces more robust

Table 6.8: Comparison on Breast_Cancer. Highest accuracies are highlighted in **boldface** for each A_k.

$\mathfrak{L}$	l	b	r	$\lvert\mathcal{T}\rvert$	A_k		
					0	1	2
RF	∞			300	**0.974**	0.907	0.825
				500	0.969	0.918	0.830
RSM-DT	∞			300	0.969	0.933	0.861
				500	0.964	0.933	0.881
FPF	8	1	100	300	**0.974**	0.912	0.711
		2	60	300	0.964	**0.943**	0.876
		3	42	294	0.954	0.938	0.897
		4	33	297	0.954	0.933	**0.902**
		5	27	297	0.948	0.928	0.897

Table 6.9: Comparison on Spam_Base. Highest accuracies are highlighted in **boldface** for each A_k.

$\mathfrak{L}$	l	b	r	$\lvert\mathcal{T}\rvert$	A_k		
					0	1	2
RF	∞			100	**0.956**	0.580	0.181
RSM-DT	∞			100	0.920	0.830	0.691
FPF	8	1	33	99	0.930	0.810	0.367
		2	20	100	0.911	**0.834**	0.716
		3	14	98	0.896	0.832	0.752
		4	11	99	0.876	0.833	0.770
		5	9	99	0.859	0.827	**0.772**

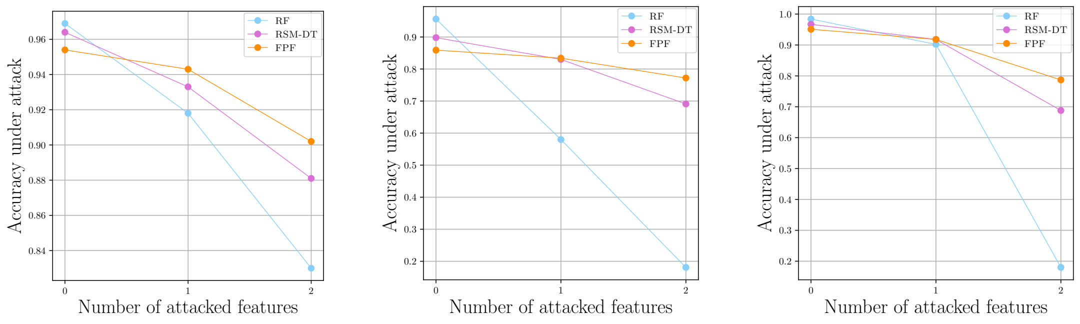
models than those produced by RSM-DT and RF, at the price of less accuracy on original instances. Finally, in general it can be seen that the more robust a model becomes, the more its accuracy on normal instances decreases.

6.3 Certificates analysis

In this last section of the chapter we exposed the results of the analyses performed on the certification methods. In the first part we showed the accuracy lower-bound calculated with ALB and FLB, and the error made with respect to BF. While in the second part we showed how through FLB it is possible to calculate the accuracy lower-bound even with very large values of b, k and r.

Table 6.10: Comparison on Binary_Wine. Highest accuracies are highlighted in **boldface** for each A_k.

$\mathfrak{L}$	l	b	r	$\lvert\mathcal{T}\rvert$	A_k		
					0	1	2
RF	∞			300	**0.984**	0.868	0.147
				1000	**0.984**	0.902	0.180
RSM-DT	∞			300	0.967	**0.918**	0.688
				1000	0.967	**0.918**	0.688
FPF	8	1	100	300	**0.984**	0.885	0.508
		2	60	300	**0.984**	**0.918**	0.705
		3	42	294	0.967	**0.918**	0.738
		4	33	297	0.967	**0.918**	0.754
		5	27	297	0.951	**0.918**	**0.787**



(a) Breast_Cancer (b) Spam_Base (c) Binary_Wine

Figure 6.2: Models comparison for each dataset.

6.3.1 Lower-bounds performance

Tables 6.11, 6.13 and 6.12 summarize respectively for Breast_Cancer, Spam_Base and Binary_Wine the accuracy lower-bound estimated by two certificates and the real robustness computed with BF. We use the notation $\Delta\%$ for the percentage error of the certificate compared to BF. For each dataset and A_1, the certificates ALB and FLB estimate the same accuracy. The highest $\Delta\%$ for each dataset is 1.6% on Breast_Cancer, 1.7% on Binary_Wine and 8% on Spam_Base. The accuracy estimated by the lower bounds is very close to the real one. With A_2 and $b = 1$, both lower bounds make a 100% error. This is normal for the following reason. With $b = 1$ the model is trained to protect only one feature. If the attacker can modify 2 features then the attacked features can be in more than half of the forest and are probably used in more than half of the decision paths. In this scenario for the assumption made by the lower-bounds: using an attacked feature in the prediction of $\boldsymbol{x}$ involves an incorrect prediction, leads to having the majority of the forest under attack, with the consequent estimation of the accuracy lower-bound equal to 0. For A_2 and small b (excluding $b = 1$), searching for all possible combinations of features of ALB turns out to be a better strategy for calculating the accuracy lower-bound, as the result is more close to the real one compared to the one estimated by FLB. For example for $b = 2$ on Binary_Wine, the algorithms get Δ%ALB = 11.6% and Δ%FLB = 27.9%. As b increases the certificates significantly improve their ability to approximate real accuracy. This is due to the fact that with large b trees are less accurate and therefore the assumption made by the lower bounds is very probable, and the lower bounds are more accurate. Furthermore, as b increases, the accuracy under attack predicted by ALB and FLB get closer and closer. This because the less features trees have, the more likely they are all used in the decision path and therefore the second approximation made by fast lower-bound [1] turns out to be as precise as the search for all combinations $C_{(d,b)}$ done by ALB. In general, both algorithms approximate an accuracy very close to the real one. This means that, the proposed lower bounds can accurately certify the non-attackability of a large portion of instances without the cost of the brute-force exploration.

Table 6.11: Lower bound analysis Breast_Cancer. FPF with $|\mathcal{T}| \approx 300$, $l = 8$.

b	A_1			A_2			A_1		A_2	
	BF	ALB	FLB	BF	ALB	FLB	Δ%ALB	Δ%FLB	Δ%ALB	Δ%FLB
1	0.912	0.897	0.897	0.711	0.0	0.0	1.6	1.6	100.0	100.0
2	0.943	0.933	0.933	0.876	0.856	0.845	1.1	1.1	2.3	3.5
3	0.938	0.928	0.928	0.897	0.876	0.876	1.1	1.1	2.3	2.3
4	0.933	0.923	0.923	0.902	0.897	0.887	1.1	1.1	0.6	1.7
5	0.928	0.928	0.928	0.897	0.892	0.887	0.0	0.0	0.6	1.1

Table 6.12: Lower bound analysis Binary_Wine. FPF with $|\mathcal{T}| \approx 300$, $l = 8$.

b	A_1			A_2			A_1		A_2	
	BF	ALB	FLB	BF	ALB	FLB	Δ%ALB	Δ%FLB	Δ%ALB	Δ%FLB
1	0.885	0.885	0.885	0.508	0.0	0.0	0.0	0.0	100.0	100.0
2	0.918	0.902	0.902	0.705	0.623	0.508	1.7	1.7	11.6	27.9
3	0.918	0.918	0.918	0.738	0.705	0.705	0.0	0.0	4.5	4.5
4	0.918	0.918	0.918	0.754	0.721	0.721	0.0	0.0	4.4	4.4
5	0.918	0.918	0.918	0.787	0.787	0.77	0.0	0.0	0.0	2.2

[1] FLB takes the k features involving the largest number of trees regardless of whether they can share trees and therefore overestimate the strength of the attacker

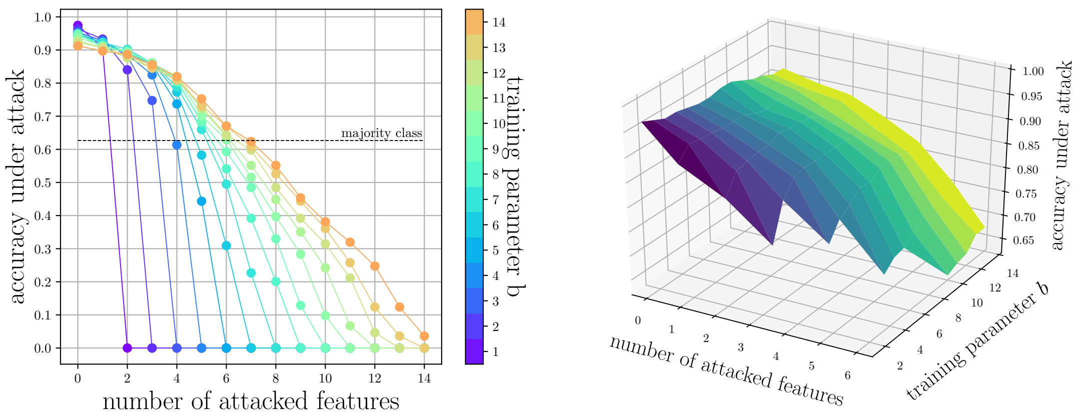
Table 6.13: Lower bound analysis Spam_Base. FPF with $|\mathcal{T}| \approx 100$, $l = 8$.

b	A_1			A_2			A_1		A_2	
	BF	ALB	FLB	BF	ALB	FLB	Δ%ALB	Δ%FLB	Δ%ALB	Δ%FLB
1	0.81	0.745	0.745	0.367	0.0	0.0	8.0	8.0	100.0	100.0
2	0.834	0.806	0.806	0.716	0.566	0.523	3.4	3.4	20.9	27.0
3	0.832	0.819	0.819	0.752	0.691	0.682	1.6	1.6	8.1	9.3
4	0.833	0.822	0.822	0.77	0.737	0.733	1.3	1.3	4.3	4.8
5	0.827	0.82	0.82	0.772	0.749	0.747	0.8	0.8	3.0	3.2

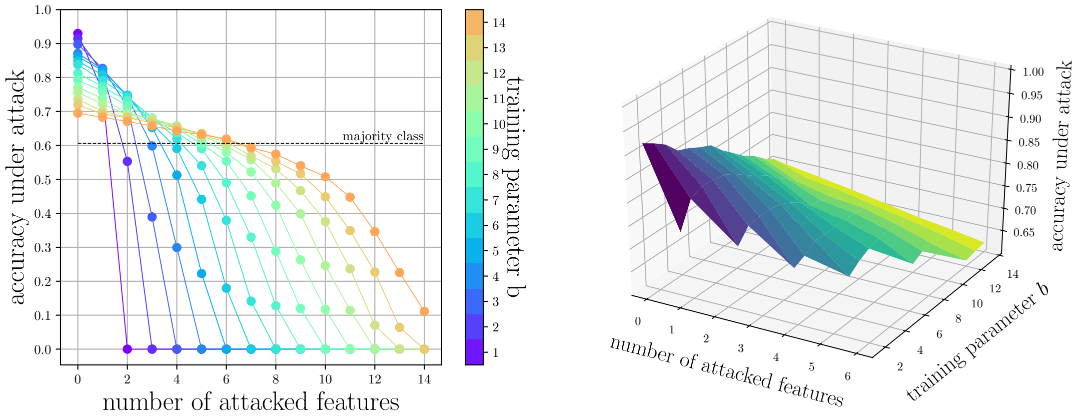
6.3.2 Fast lower-bound utility

In Figure 6.3 and 6.4 we showed the accuracy lower-bound computed with FLB on different dataset, by varying b and the attacker budget k. For each dataset we provided a 2D and 3D version of the performance of the models under attack. The 2D plot highlights the accuracy obtained by the models with the variation of k. Instead the 3D one allows to have a better view of the trend of the Acc_{A_k} as b increases. For 3D plots, values below the majority class threshold are cut for better view. The polynomial temporal complexity of fast lower-bound allows to compute the accuracy lower-bound for large values of b, k and r. The figure shows how larger values of b allow to sustain a larger attacker strength. Of course, when the attacker becomes too strong compared with the number of relevant features in the dataset the accuracy of FPF drops. As mentioned in dataset section 5.1.1, in this experiment we also included three datasets generated from MNIST by isolating instances of two digits. The results for Breast_Cancer dataset are shown in Figures 6.3a. With FPF it is possible to train a model that has an accuracy lower-bound of over 0.85 attacked by A_3, an accuracy greater than RF attacked by A_2. With A_4 it's possible to train a model with accuracy lower-bound of roughly 0.82, slightly below RF with A_2. As the attacker gets stronger, the degradation of the model's performance is inevitable. For Spam_Base dataset, in Figures 6.3b the phenomenon mentioned in the evaluation methodology section 5.3.5 about the parameter b is particularly evident. As b grows individual trees become less accurate and robustness decreases on weak attacks. On larger attacks, obviously a large b has better performances. For Binary_Wine dataset, in Figures 6.3c it can be seen that an attacker with budget $k > 2$ leads to a degradation of performances below the majority class threshold. Predictable behavior, considering that Binary_Wine has only 13 features and only 7 are relevant. To protect 3 features FPF generates 7 trees for each round and each tree has 2 features. In the best case, each tree has an important feature. Expecting single trees to be robust to such a strong attack is unreasonable. The attack L_0 is very strong so even the attack of a few features can make the difference. For the datasets deriving from MNIST we wanted

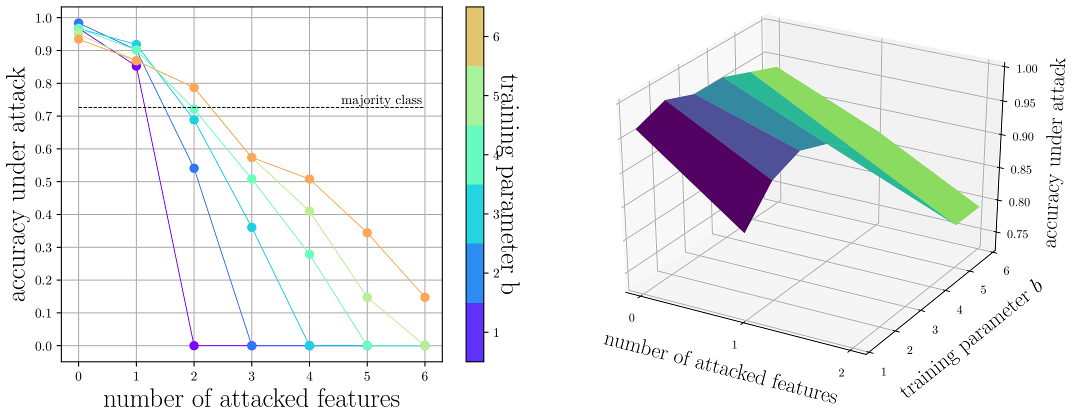
to show how FLB can calculate the accuracy lower-bound also with a number of instances greater than 15000 each with 784 features and for each budget $k \leq 100$. The Figures 6.4a of the analysis on MNIST 0/1 dataset, show how it is possible to train a model robust to an attack on 20 features and still have an accuracy under attack of roughly 0.97. With a stronger attacker for example A_{30} or A_{40} it is possible to guarantee a lower-bound accuracy of 90% and 73% respectively. Considering that the important features are 56 the results are remarkable. For MNIST 1/7 dataset, reported in Figures 6.4b the performances degrade earlier than the previous dataset. It is not difficult to imagine why. The number of pixels to be changed to make a 1 look like a 7 is less than needed to turn a 0 into a 1. fast lower-bound estimated an accuracy lower-bound of 95% and 83% attacking respectively 10 and 20 features. Finally, the results for the MNIST 5/6 dataset, reported in Figures 6.4c, show how the performances degrade even much faster than the previous ones. The reason is the same as before, but much more evident. The digits 5 and 6 are very similar, and the attacker can simply modify a few pixels to deceive the model. By modifying 10 pixels the model correctly classifies the 90% of the instances and attacking 20 features the 74%. Obviously these are accuracy lower-bound so the actual accuracy under attack could be higher than these, especially since, as we have seen in the previous section, the lower-bounds error grows with k.
Finally, the 3D plots suggest that as k increases, models that maintain accuracy under attack above the majority class threshold are those that have a high b. As k grows, b must also grow. Concluding, with this experiment we have shown how FLB can be used to give an accurate lower-bound of the model's performances as k grows, which would be infeasible with the use of the BF algorithm. We showed the power of FLB both for large values of b and k but also for large values of $|\mathcal{T}|$, reaching up to 3000 trees per model. The presence of many trees in the forest involves many different thresholds for each feature and therefore increases the execution time of BF. For FLB the increase in the number of trees is extremely less incisive. With FLB it is possible to quickly train many models for different values of the training parameters and understand which are the best values to create the most robust model against A_k. In addition, it can be used to filter non-attackable instances and significantly reduce the execution time of ALB and BF.



(a) Breast_Cancer, $|\mathcal{T}| \approx 1000$, $l = 8$.

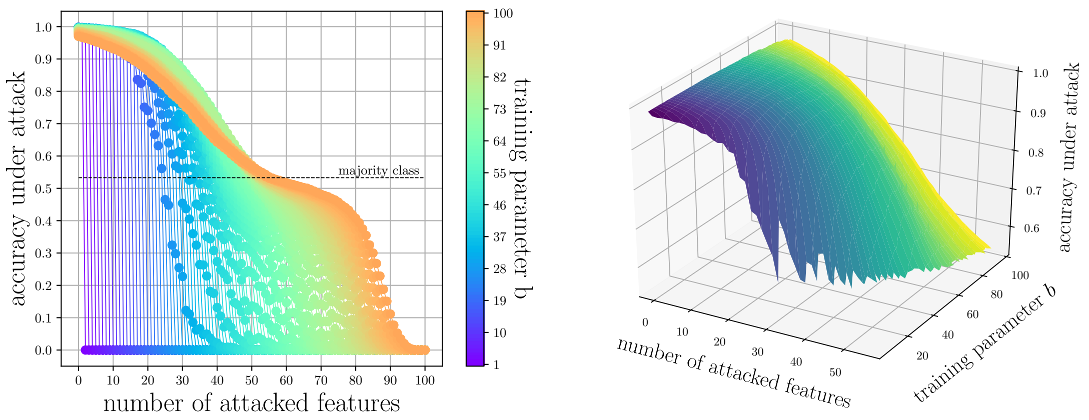


(b) Spam_Base, $|\mathcal{T}| \approx 1000$, $l = 8$.

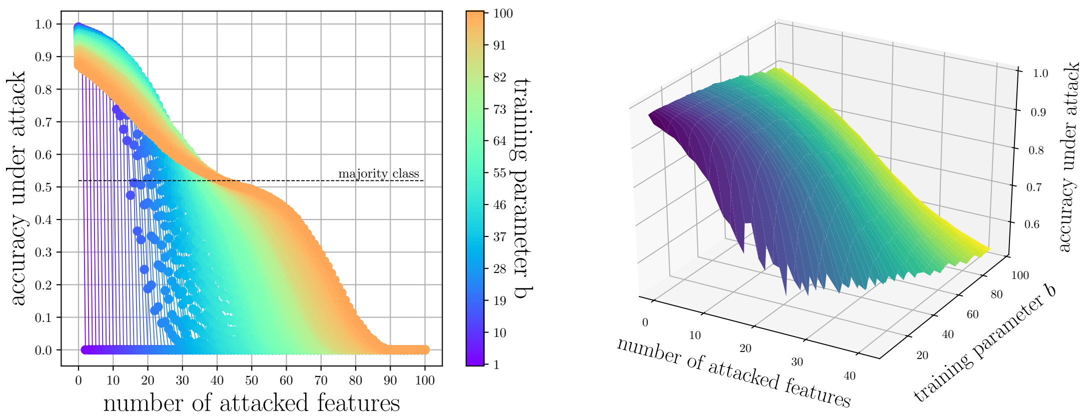


(c) Binary_Wine, $|\mathcal{T}| \approx 3000$, $l = 8$.

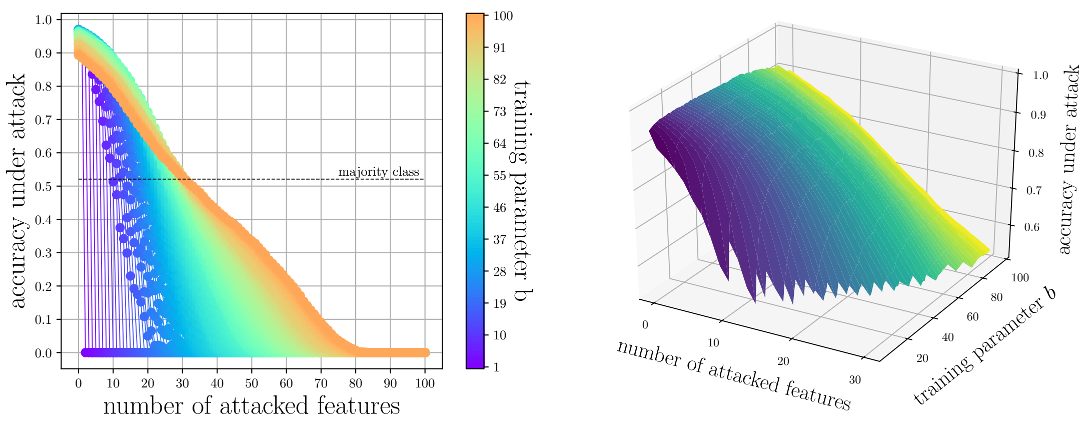
Figure 6.3: Accuracy under attack computed with fast lower-bound by varying b and k.



(a) MNIST 0/1 $|\mathcal{T}| \approx 500$, $l = 8$.



(b) MNIST 1/7 $|\mathcal{T}| \approx 500$, $l = 8$.



(c) MNIST 5/6 $|\mathcal{T}| \approx 500$, $l = 8$.

Figure 6.4: Accuracy under attack computed with fast lower-bound by varying b and k.

Summary

In this chapter we collected all the results of the experiments defined in the previous chapter. We have summarized the main sections of the chapter below.

- **Parameters analysis.** For each algorithm: **FPF**, **RF** and **RSM-DT** we discussed the results performed for each training parameter on **Breast_Cancer** dataset. We have seen that for FPF models the best maximum number of leaves is $l = 8$. For the parameter r we have seen that in general the robustness of the ensemble increases with the size of the forest. For this reason, we trained the model with r large enough to be robust.Finally, as b increases, a model becomes more robust with stronger attacks, but it loses accuracy on original instances and attacks with smaller budgets. For smaller b the model acquires greater robustness as the forest grows. For the RSM-DT algorithm we have seen that the size of the subset that generates the most robust models against A_k is obtained with $p = 0.2$, while the best maximum number of leaves is $l = \infty$. Finally, the forest size analysis showed that both RF and RSM-DT gain robustness against A_k as $|\mathcal{T}|$ increases.

- **Algorithms comparison.** Through the best values of the training parameters found with the experiments, we have trained for each dataset **Breast_Cancer**, **Spam_Base** and **Binary_Wine** the models used in the comparisons. From the comparisons we have seen that, for each dataset, **Feature Partitioned Forest** has greater robustness than RSM-DT and RF for attacks generated by A_1 and A_2.

- **Certificates analysis.** Finally we discussed the results of the experiments performed on the lower bounds. In particular we showed that both **ALB** and **FLB** approximte the accuracy under attack very close to the real one calculated with BruteForce. We have seen that accurate lower-bound performs better with small b and when b increases ALB and FLB perform similarly. Finally we showed how through FLB it is possible to calculate a lower bound of the performance of FPF models even with very large k, b and r. Calculating the real accuracy under attack with the parameters used in the experiments is infeasible with BruteForce.

Chapter 7

Conclusion

It is known traditional machine learning algorithms are vulnerable to attacks perpetuated by an intelligent agent called adversary. The attacks can be of various types and affect both the training phase and the operability phase of the model. Among all the types of attack, we focused on evasion attacks and defense strategies for this type of vulnerability. In an evasion attack, the adversary can perturb with a certain budget k, an instance $\boldsymbol{x} \in \mathcal{X}$ such that the resulting evading instance $\boldsymbol{x}'$ produces $\mathcal{T}(\boldsymbol{x}) \neq \mathcal{T}(\boldsymbol{x}')$. In literature there are various strategies to contrast this type of attack, designed for the most common machine learning algorithm. At the same time, only a small part of them focuses on decision trees and ensemble methods based on them.

In this thesis, we proposed Feature Partitioned Forest, a new ensemble learning algorithm for the generation of decision tree forest, robust to evasion attack such that $\| \boldsymbol{x} - \boldsymbol{x}' \|_0 \leq k$. The value of k is called budget and it specifies the strength of the attacker A_k. This means that the evading instance $\boldsymbol{x}'$ can differ from $\boldsymbol{x}$ at most on k features. The FPF algorithm that we proposed is based on a robust partitioning of the set of features $\mathcal{F}$. To counteract an evasion attack performed on b features, the FPF algorithm robustly partitions the set of features into $2b + 1$ disjoint subsets. Subsequently, for each partition, a projection of the dataset is performed on which a binary tree is trained. This generates a forest of $2b + 1$ trees, whose features have an empty intersection. To increase the robustness of the model, FPF repeats r times the robust partitioning on a random order of features to generate other robust partitions, different from the previous ones, and trains $2b + 1$ trees at each round. In total, the ensemble produced by FPF contains $r(2b + 1)$ trees. Thanks to this robust partitioning of the feature set, given an attacker A_k with a budget $k \leq b$, the evasion instances generated involve at most rk trees, less than half of the forest, so the ensemble does not change its prediction.

In addition we provided two certificates to efficiently calculate a lower bound of the performance that the model obtains under attack, which experimentaly proved to be very accurately. The approximation performed by the lower bounds is based on the assumption that trees that uses a feature attacked by A_k, in the prediction of $\boldsymbol{x}$, commits a misclassification of $\boldsymbol{x}'$. The first certificate is called fast lower-bound, whose name derives from its polynomial complexity. The algorithm approximates the robustness of the model by looking for the k features inside $\mathcal{T}$ that break the largest number of trees. If the number of trees that correctly predict the label is greater than the number of trees that make a mistake/break, then the ensemble is robust for every attack on k features. This lower bound can overestimate the attacker's strength when the chosen k features share some trees and therefore these trees would be counted multiple times To improve the accuracy of the lower bound we also developed the accurate lower-bound which predicts the robustness more precisely than the previous algorithm but with higher computational cost. The ALB calculates all combinations of $C_{(|\mathcal{F}|,k)}$ of attackable features and for each one looks at which trees are involved in the attack. If most of the trees predict the correct label then the combination of features is not an attack. If the forest is robust to every possible combination of features then there is no attack of A_k that evades the model.

The real robustness of the model can be calculated with a brute-force strategy that searches for all possible perturbations of $\boldsymbol{x}$ on $\mathcal{T}$. However, this strategy has a prohibitive cost as the forest size and budget k grow, and as we said before, computing all possible evading instances with a brute-force strategy is NP-complete. Through the experiments conducted we have shown how the certificates can very precisely approximate the real robustness of the model, at a significantly lower computational cost than the brute-force strategy.

Finally we showed that FPF is more robust against A_k than ensembles produced by Random Forest and a state-of-the-art algorithm Random Subspace Method. For each algorithm, we performed an extensive analysis of the training parameters to better train each model and have a more fair comparison. In general, we have seen that as the forest grows, each model acquires greater robustness to attacks. Furthermore, we have shown empirically how the maximum number of leaves in each tree affects the robustness of the ensemble. Concluding FPF is proved to be resilient against evasion attacks, and, more importantly, we are able to certificate in a very efficient way that, given a test dataset, some of the instances cannot be attacked at all, thus avoiding the costly computation of all the possible evasion attacks.

From this results we conclude that FPF is able to outperform state-of-the-art competitors especially with a stronger attacker. FPF achieves an accuracy under attack of 12.6% and 10.5% higher than that obtained from RSM on Spam_Base and Binary_Wine datasets. Instead, compared to RF, the algorithm we have proposed has a 76.5 and 81.3 percent increase in accuracy under attack, rispectively for

Spam_Base and Binary_Wine datasets. The experimental evaluation, carried out on publicly available datasets, is promising and ouperforms the main direct competitor RSM-DT, based on ensembles build on random sampling of the features. Moreover, we also show that our certified lower bounds on the accuracy under attack are significant.

Chapter 8

Future Works

In this last chapter we have collected some ideas of future works, starting from this thesis. As we have seen FPF has an important limitation, in fact, the algorithm was developed to be applied only in binary classification scenarios. As future work we want to investigate the extension of the model from binary classification model to multi-class classification model and investigate if the same properties and certifications can still be maintained. Feature Partitioned Forest was developed and tested using decision trees as weak-learners, but the robustness properties by construction are not related to this specific learning algorithm. Thus another point of research is to use FPF with other base-learners. The proposed certificates are closely related to the structure of the decision tree, so it is necessary to investigate whether it is possible to recreate the same type of certificates with different base-learners. GBDT is the state-of-the-art of tree-based models in non-adversarial contexts. It would be interesting to implement FPF robust partition with GBDT as a base-learner. Given the dependence that each tree in GBDT has on its successor, a future work could be to investigate whether it is possible to train a possile FPF-GBDT so that the attacker can still involve only less than half of the forest when performing an attack. As was said in the chapter 4, one of the reasons why FPF badly classifies some instances under attack is that some trees make mistakes even without being attacked, while the attacked trees break because they are not robust taken individually. In this regard, it would be interesting to combine the ensemble learning algorithm FPF with robust base-learners against adversarial examples and see if the performance improves. For example, decision trees generated with *Treant* algorithm shown in section 3.5 can be used as robust base-learners. Finally, FPF creates a robust model for evasion attacks modeled with distance function L_0. Another future work is to investigate whether the combination of FPF and robust base-leaners against adversarial examples constrained by L_1-norm, L_2-norm and L_∞-norm, improve performance against their adversary's model.

Bibliography

[1] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. “Can machine learning be secure?” In: *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2006, Taipei, Taiwan, March 21-24, 2006.* ACM, 2006, pp. 16–25. DOI: 10.1145/1128817.1128824.

[2] Battista Biggio, Igino Corona, Giorgio Fumera, Giorgio Giacinto, and Fabio Roli. “Bagging Classifiers for Fighting Poisoning Attacks in Adversarial Classification Tasks”. In: *Multiple Classifier Systems - 10th International Workshop, MCS 2011, Naples, Italy, June 15-17, 2011. Proceedings.* Vol. 6713. Lecture Notes in Computer Science. Springer, 2011, pp. 350–359. DOI: 10.1007/978-3-642-21557-5_37.

[3] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Srndic, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. “Evasion Attacks against Machine Learning at Test Time”. In: *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III.* Vol. 8190. Lecture Notes in Computer Science. Springer, 2013, pp. 387–402. DOI: 10.1007/978-3-642-40994-3_25.

[4] Battista Biggio, Igino Corona, Blaine Nelson, Benjamin I. P. Rubinstein, Davide Maiorca, Giorgio Fumera, Giorgio Giacinto, and Fabio Roli. “Security Evaluation of Support Vector Machines in Adversarial Environments”. In: *CoRR* abs/1401.7727 (2014). arXiv: 1401.7727.

[5] Battista Biggio, Giorgio Fumera, and Fabio Roli. “Multiple classifier systems for robust classifier design in adversarial environments”. In: *Int. J. Machine Learning & Cybernetics* 1.1-4 (2010), pp. 27–41. DOI: 10.1007/s13042-010-0007-7.

[6] Battista Biggio, Giorgio Fumera, and Fabio Roli. “Multiple Classifier Systems under Attack”. In: *Multiple Classifier Systems, 9th International Workshop, MCS 2010, Cairo, Egypt, April 7-9, 2010. Proceedings.* Ed. by Neamat El

Gayar, Josef Kittler, and Fabio Roli. Vol. 5997. Lecture Notes in Computer Science. Springer, 2010, pp. 74–83. DOI: 10.1007/978-3-642-12127-2_8.

[7] Battista Biggio, Giorgio Fumera, and Fabio Roli. "Security Evaluation of Pattern Classifiers under Attack". In: *CoRR* abs/1709.00609 (2017). arXiv: 1709.00609.

[8] Battista Biggio and Fabio Roli. "Wild patterns: Ten years after the rise of adversarial machine learning". In: *Pattern Recognition* 84 (2018), pp. 317–331. DOI: 10.1016/j.patcog.2018.07.023.

[9] Leo Breiman. "Bagging Predictors". In: *Machine Learning* 24.2 (1996), pp. 123–140. DOI: 10.1007/BF00058655.

[10] Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32. DOI: 10.1023/A:1010933404324.

[11] Qi-Zhi Cai, Chang Liu, and Dawn Song. "Curriculum Adversarial Training". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.* ijcai.org, 2018, pp. 3740–3747. DOI: 10.24963/ijcai.2018/520.

[12] Stefano Calzavara, Claudio Lucchese, and Gabriele Tolomei. "Adversarial Training of Gradient-Boosted Decision Trees". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019.* ACM, 2019, pp. 2429–2432. DOI: 10.1145/3357384.3358149.

[13] Hongge Chen, Huan Zhang, Duane S. Boning, and Cho-Jui Hsieh. "Robust Decision Trees Against Adversarial Examples". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 1122–1131.

[14] Hongge Chen, Huan Zhang, Si Si, Yang Li, Duane S. Boning, and Cho-Jui Hsieh. "Robustness Verification of Tree-based Models". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada.* Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett. 2019, pp. 12317–12328.

[15] Yan Chen, Wei Wang, and Xiangliang Zhang. "Randomizing SVM Against Adversarial Attacks Under Uncertainty". In: *Advances in Knowledge Discovery and Data Mining - 22nd Pacific-Asia Conference, PAKDD 2018, Melbourne, VIC, Australia, June 3-6, 2018, Proceedings, Part III.* Ed. by Dinh Q. Phung, Vincent S. Tseng, Geoffrey I. Webb, Bao Ho, Mohadeseh

Ganji, and Lida Rashidi. Vol. 10939. Lecture Notes in Computer Science. Springer, 2018, pp. 556–568. DOI: 10.1007/978-3-319-93040-4_44.

[16] Minhao Cheng, Thong Le, Pin-Yu Chen, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. "Query-Efficient Hard-label Black-box Attack: An Optimization-based Approach". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[17] Corinna Cortes and Vladimir Vapnik. "Support-Vector Networks". In: *Machine Learning* 20.3 (1995), pp. 273–297. DOI: 10.1007/BF00994018.

[18] Nilesh N. Dalvi, Pedro M. Domingos, Mausam, Sumit K. Sanghai, and Deepak Verma. "Adversarial classification". In: *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*. ACM, 2004, pp. 99–108. DOI: 10.1145/1014052.1014066.

[19] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017.

[20] Bradley Efron and Robert Tibshirani. *An Introduction to the Bootstrap*. Springer, 1993. ISBN: 978-1-4899-4541-9. DOI: 10.1007/978-1-4899-4541-9.

[21] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. "Robust Physical-World Attacks on Deep Learning Visual Classification". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. IEEE Computer Society, 2018, pp. 1625–1634. DOI: 10.1109/CVPR.2018.00175.

[22] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. "Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*. ACM, 2015, pp. 1322–1333. DOI: 10.1145/2810103.2813677.

[23] Jerome H. Friedman. "Greedy Function Approximation: A Gradient Boosting Machine". In: *Annals of Statistics* 29 (2000), pp. 1189–1232.

[24] Jerome H. Friedman. "Stochastic Gradient Boosting". In: *Comput. Stat. Data Anal.* 38.4 (2002), pp. 367–378. ISSN: 0167-9473. DOI: 10.1016/S0167-9473(01)00065-2.

[25] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015.

[26] Yves Grandvalet. “Bagging Equalizes Influence”. In: *Machine Learning* 55.3 (2004), pp. 251–270. DOI: `10.1023/B:MACH.0000027783.34431.42`.

[27] Warren He, James Wei, Xinyun Chen, Nicholas Carlini, and Dawn Song. “Adversarial Example Defense: Ensembles of Weak Defenses are not Strong”. In: *11th USENIX Workshop on Offensive Technologies, WOOT 2017, Vancouver, BC, Canada, August 14-15, 2017.* USENIX Association, 2017.

[28] Tin Kam Ho. “Random decision forests”. In: *Third International Conference on Document Analysis and Recognition, ICDAR 1995, August 14 - 15, 1995, Montreal, Canada. Volume I.* IEEE Computer Society, 1995, pp. 278–282. DOI: `10.1109/ICDAR.1995.598994`.

[29] Tin Kam Ho. “The Random Subspace Method for Constructing Decision Forests”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 20.8 (1998), pp. 832–844. DOI: `10.1109/34.709601`.

[30] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I. P. Rubinstein, and J. D. Tygar. “Adversarial machine learning”. In: *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence, AISec 2011, Chicago, IL, USA, October 21, 2011.* ACM, 2011, pp. 43–58. DOI: `10.1145/2046684.2046692`.

[31] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. “Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning”. In: *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, 2018, pp. 19–35. DOI: `10.1109/SP.2018.00057`.

[32] Alex Kantchelian, J. D. Tygar, and Anthony D. Joseph. “Evasion and Hardening of Tree Ensemble Classifiers”. In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016.* Vol. 48. JMLR Workshop and Conference Proceedings. JMLR.org, 2016, pp. 2387–2396.

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

[34] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. “Adversarial Machine Learning at Scale”. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings.* OpenReview.net, 2017.

[35] Yann LeCun and Corinna Cortes. “MNIST handwritten digit database”. In: (2010).

[36] Daniel Lowd and Christopher Meek. “Adversarial learning”. In: *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*. ACM, 2005, pp. 641–647. DOI: 10.1145/1081870.1081950.

[37] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. “Towards Deep Learning Models Resistant to Adversarial Attacks”. In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[38] Marco Melis, Ambra Demontis, Battista Biggio, Gavin Brown, Giorgio Fumera, and Fabio Roli. “Is Deep Learning Safe for Robot Vision? Adversarial Examples Against the iCub Humanoid”. In: *2017 IEEE International Conference on Computer Vision Workshops, ICCV Workshops 2017, Venice, Italy, October 22-29, 2017*. IEEE Computer Society, 2017, pp. 751–759. DOI: 10.1109/ICCVW.2017.94.

[39] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C. Lupu, and Fabio Roli. “Towards Poisoning of Deep Learning Algorithms with Back-gradient Optimization”. In: *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2017, Dallas, TX, USA, November 3, 2017*. ACM, 2017, pp. 27–38. DOI: 10.1145/3128572.3140451.

[40] Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D. Joseph, Benjamin I. P. Rubinstein, Udam Saini, Charles A. Sutton, J. Doug Tygar, and Kai Xia. “Exploiting Machine Learning to Subvert Your Spam Filter”. In: *First USENIX Workshop on Large-Scale Exploits and Emergent Threats, LEET ’08, San Francisco, CA, USA, April 15, 2008, Proceedings*. USENIX Association, 2008.

[41] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. “A General Framework for Adversarial Examples with Objectives”. In: *ACM Trans. Priv. Secur.* 22.3 (2019), 16:1–16:30. DOI: 10.1145/3317611.

[42] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015.

[43] Marina Skurichina and Robert P. W. Duin. “Bagging, Boosting and the Random Subspace Method for Linear Classifiers”. In: *Pattern Anal. Appl.* 5.2 (2002), pp. 121–135. DOI: 10.1007/s100440200011.

[44] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. “One Pixel Attack for Fooling Deep Neural Networks”. In: *IEEE Trans. Evolutionary Computation* 23.5 (2019), pp. 828–841. DOI: `10.1109/TEVC.2019.2890858`.

[45] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. “Intriguing properties of neural networks”. In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings.* 2014.

[46] Dacheng Tao, Xiaoou Tang, Xuelong Li, and Xindong Wu. “Asymmetric Bagging and Random Subspace for Support Vector Machines-Based Relevance Feedback in Image Retrieval”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 28.7 (2006), pp. 1088–1099. DOI: `10.1109/TPAMI.2006.134`.

[47] Vladimir Vapnik. “Principles of Risk Minimization for Learning Theory”. In: *Advances in Neural Information Processing Systems 4, [NIPS Conference, Denver, Colorado, USA, December 2-5, 1991].* Morgan Kaufmann, 1991, pp. 831–838.

[48] Z.H. Zhou. *Ensemble Methods: Foundations and Algorithms.* Chapman & Hall/CRC Machine Learning & Pattern Recognition Series. Taylor & Francis, 2012. ISBN: 9781439830031.

Ringraziamenti

In quest'ultima parte sento la necessità di ringraziare tutte le persone che mi hanno reso possibile il raggiungimento di questo importante traguardo.

Il ringraziamento più sentito e sincero va al mio relatore, il professor Claudio Lucchese. Impossibile ringraziarlo abbastanza per tutto il tempo che ha saputo dedicarmi nello sviluppo di questa tesi, per l'aiuto, gli insegnamenti, e le opportunità che mi ha dato. Lo ringrazio per l'infinita pazienza e per aver creduto in me più di quanto lo abbia fatto io. Infine lo ringrazio per avermi dato l'occasione di lavorare ad un progetto stimolante a cui mi sono appassionato.

Ringrazio calorosamente la mia famiglia senza la quale non avrei mai potuto raggiungere questo obiettivo e tutte le persone importanti che mi hanno sostenuto in questi anni.

Un doveroso ringraziamento va a tutti i miei compagni di corso, che negli anni sono diventati gli amici più cari. Non li ringrazierò mai abbastanza per la bellissima esperienza universitaria che mi hanno permesso di vivere.

Infine ringrazio chiunque abbia dedicato del tempo a leggere questa tesi.

Acknowlegments

I want to thank the people who have supported me in achieving this important goal.

The most sincere thanks go to my supervisor, Professor Claudio Lucchese. It is impossible to thank him enough for all the time he has dedicated to me in developing this thesis, for the help, the teachings, and the opportunities he has given me. I thank him for his infinite patience and for believing in me more than I did. Finally, I thank him for giving me the opportunity to work on this stimulating project.

I thank my family without whom I could never have achieved this goal and all the important people who have supported me over the years.

A heartfelt thanks goes to all my classmates, who over the years have become the closest friends. I can never thank them enough for the wonderful university experience I have had.

Finally, I thank anyone who read this thesis.