Tampereen yliopisto

Onni Mansikkamäki

ROBUST DECISION TREES UNDER ADVERSARIAL ATTACKS

Bachelor of Science Thesis
Faculty of Information Technology and Communication Sciences
Examiner: Tapio Elomaa
May 2022

ABSTRACT

Onni Mansikkamäki: Robust decision trees under adversarial attacks
Bachelor of Science Thesis
Tampere University
Electrical Engineering
May 2022

Normal decision trees are effective but simple machine learning models that are prone to adversarial attacks. Nevertheless, the operation of decision trees under adversarial attacks has received relatively little research, and robust decision tree algorithms that can withstand these attacks have only been developed in recent years. The purpose of this work is to determine how accurately, robustly, and time-efficiently different robust decision tree models perform under the attack compared to each other, and how accurately they perform compared to non-robust decision tree models under attack.

Adversarial attacks create adversarial examples, that allow the attacker to try and affect the decision tree's ability to perform accurately in given classification task. Adversarial examples are data samples that are designed to appear normal to the ordinary decision tree, and whose numerical properties have been modified to trick the decision tree into making an incorrect classification decision. Robust decision trees are decision tree models designed to withstand adversarial attacks.

The work first introduces the essential backgrounds of the topic of the work. The background covers machine learning, supervised learning, decision trees and tree ensembles, and learning procedure of a robust decision tree model. Once the necessary background information is covered, methods as well as topics related to the research phase are reviewed, which will allow the performance of decision trees as well as robust decision trees to be examined under adversarial attacks. The research part of the thesis presents the parts of the research and finally the results produced.

In the research part of the thesis, source code retrieved from GitHub is executed. The source code used 16 different datasets with two different result categories. The datasets were retrieved from the OpenML. There were 14 datasets of structural datasets, and the other 2 datasets were image datasets. Only binary classification tasks were assigned to the studied trees and tree ensembles. The source code performed attacks, built trees and tree ensembles, and produced result images related to the accuracy, robustness, and time efficiency of the models.

The results produced by the research part of the thesis show, that all robust decision tree models used in the study perform classification much more accurately under attack, than other non-robust decision tree models used in the study. The results show that the GROOT tree, which performed by two percentage points to the best decision tree in accuracy, was the most time efficient robust decision tree for almost all datasets. The results show that of the robust decision tree ensembles, GROOT forest, which performs the best in terms of prediction accuracy, was the robust decision tree ensemble that performs the best in terms of time efficiency for almost all datasets. The results also shows that for the image datasets, the robust decision tree models that participated in the robustness part of the study were almost equally robust for all images in both image datasets.

Keywords: machine learning, decision trees, robust decision trees, adversarial attacks

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

TIIVISTELMÄ

Onni Mansikkamäki: Sietokykyiset päätöspuut vihamielisten hyökkäysten kohteena
Kandidaatintyö
Tampereen yliopisto
Sähkötekniikka
Toukokuu 2022

Normaalit päätöspuut ovat tehokkaita, mutta yksinkertaisia koneoppimismalleja, jotka ovat alttiita vihamielisille hyökkäyksille (*adversarial attack*). Tästä huolimatta päätöspuiden toimintaa vihamielisten hyökkäysten alaisena on tutkittu verrattain vähän ja hyökkäyksiä kestäviä sietokykyisiä (*robust*) päätöspuualgoritmeja on kehitetty enenevissä määrin vasta viime vuosina. Tämän työn tarkoituksena on selvittää, kuinka tarkasti, sietokykyisesti sekä aikatehokkaasti erilaiset sietokykyiset päätöspuumallit suoriutuvat vihamielisten hyökkäysten alaisena verrattuna toisiinsa, sekä kuinka tarkasti ne suoriutuvat verrattuna ei-sietokykyisiin päätöspuumalleihin vihamielisten hyökkäysten alaisena.

Vihamieliset hyökkäykset luovat vihamielisiä datanäytteitä (*adversarial example*), joiden avulla hyökkäys yrittää vaikuttaa päätöspuun kykyyn suoriutua tarkasti luokittelutehtävästä. Vihamieliset datanäytteet ovat suunniteltuja, usein ihmiselle sekä tavalliselle päätöspuulle normaalilta vaikuttavia datanäytteitä, joiden numeerisia ominaisuuksia on muokattu niin, että ne huijaavat päätöspuuta antamaan väärän luokittelupäätöksen. Sietokykyiset päätöspuut ovat päätöspuumalleja, jotka on suunniteltu kestämään vihamielisiä hyökkäyksiä.

Työssä perehdytään ensin työn aiheen oleellisiin taustoihin. Taustoista käsitellään koneoppimista, ohjattua oppimista, päätöspuuyhdistelmiä (*tree ensemble*) sekä sietokykyisen päätöspuumallin opettamista. Kun tarvittava taustatieto on käsitelty, käydään läpi metodeja, sekä tutkimusvaiheeseen liittyviä aiheita, joiden avulla pystytään tutkimaan päätöspuiden sekä sietokykyisten päätöspuiden toimintaa vihamielisten hyökkäysten alaisena. Työn tutkimusosuudessa esitellään tutkimuksen vaiheet ja lopulta työn tutkimuksen tuottamat tulokset.

Työn tutkimuksessa ajettiin GitHubista haettua lähdekoodia. Lähdekoodi käytti 16 erilaista tietoaineistoa (*dataset*), joissa oli kaksi eri tulosluokkaa. Tietoaineistot haettiin OpenML-järjestelmästä. 14 tietoaineistoa olivat rakenteellisia tietoaineistoja (*structured dataset*) ja loput 2 tietoaineistoa olivat kuvatietoaineistoja. Tutkituille puille sekä puuyhdistelmille kohdistettiin ainoastaan binäärisiä luokittelutehtäviä. Lähdekoodi suoritti hyökkäykset, opetti ja rakensi puu- ja puuyhdistelmämallit sekä tuotti mallien tarkkuuteen, sietokykyyn sekä aikatehokkuuteen liittyviä tuloskuvia.

Työn tutkimusosuuden tuottamat tulokset näyttävät, että kaikki tutkimuksessa käytetyt sietokykyiset päätöspuumallit suoriutuvat luokittelusta vihamielisen hyökkäyksen alaisena huomattavasti tarkemmin kuin muut tutkimuksessa käytetyt ei-sietokykyiset päätöspuumallit. Tuloksista nähdään, että kahden prosenttiyksikön ennustamistarkkuuden erolla parhaimpaan päätöspuuhun suoriutunut GROOT tree oli lähes kaikkien tietoaineistojen kohdalla aikatehokkuudeltaan parhaiten suoriutunut sietokykyinen päätöspuu. Tuloksista nähtiin, että sietokykyisistä päätöspuuyhdistelmistä ennustamistarkkuuden kannalta parhaiten suoriutunut GROOT forest, oli useimpien tietoaineistojen kohdalla aikatehokkuudeltaan parhaiten suoriutunut sietokykyinen päätöspuuyhdistelmä. Tuloksista nähtiin myös, että kuvatietoaineistojen kohdalla sietokykytarkasteluun osallistuneet sietokykyiset päätöspuumallit olivat molempien kuvatietoaineistojen kaikkien kuvien kohdalla lähes yhtä sietokykyisiä.

Avainsanat: koneoppiminen, päätöspuut, sietokykyiset päätöspuut, vihamieliset hyökkäykset

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck –ohjelmalla.

PREFACE

This bachelor's thesis was written at Tampere University for the degree program in electrical engineering. I would like to thank my supervisor Prof. Tapio Elomaa for the guidance and for coming up with this topic in the first place. I also want to thank people, who organized the bachelor's thesis seminar in a way, that gave me the possibility to attend the seminar and classes remotely and so gave me the opportunity to live under the sun of Canary Islands while writing this thesis.

Tampereella, 26.5.2022

Onni Mansikkamäki

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

DT	Decision Tree
ML	Machine Learning
RF	Random Forest
ℓ_P	p-norm
$S(\cdot)$	Score function
$RS(\cdot)$	Robust score function

1. INTRODUCTION

As machine learning (ML) systems are becoming more and more common in many different industries, their ability to withstand potential attacks while maintaining their accuracy is emphasized. Especially in security, finance and health-focused industries, the weakness of ML systems to defend against different attacks can become costly and harmful to users of such system. While the accuracy and performance of the ML systems evolve, so does effectiveness of the attacks.

One of these attacks is often invisible, especially to the human eye as well as normal ML system. *Adversarial attacks* generate *adversarial examples* which, on the outside, appear to be completely normal instances in data, but are designed to force the system to misclassify the samples. *Adversarial learning* is used against adversarial attacks, with the objective to train attack-resistant models. In other words, adversarial learning improves model's *robustness,* ability to handle and tolerate samples that are designed to fool the model and force false prediction. [1]

Although the effects of adversarial examples on ML implementations such as neural networks and deep learning have been extensively studied, the topic of adversarial examples is relatively new for decision trees (DT) [2]. The performance of DT system against adversarial examples was only first studied by Papernot et al. [3] in 2016. One of the first algorithms for training robust decision trees was introduced by Chen et al. [2] in 2019. Since 2019, different algorithms for robust DT learning have been presented and research on the subject has increasingly appeared.

1.1 Objectives of the study

In this thesis, DTs, DT ensembles, their structure, and procedure are introduced. This thesis presents how, and what kind of attacks DTs are exposed, and above all, how DTs can learn to become robust against such attacks.

The research question is divided into two parts:

1. How well state-of-the-art robust DTs and tree ensembles perform under adversarial attacks when it comes to prediction accuracy, robustness and efficiency, and how do they perform relative to each other?

2. How much more accurately the robust DTs and tree ensembles perform under adversarial attacks compared to non-robust DTs and tree ensembles?

These questions are being answered by reviewing and analyzing related work in the field, covering necessary background for understanding the topics, and reproducing an experiment, from which the final results can be obtained and later discussed.

2. BACKGROUND AND RELATED WORK

Before we can find answers to our research questions, it is necessary to go through essential background for main topics of this thesis. Section 2.1 introduces machine learning, Section 2.2 goes through supervised learning, Section 2.3 covers DTs and tree ensembles and Section 2.4 reviews robust DTs.

2.1 Machine learning

Machine learning refers to the computer implemented algorithms that, by learning from input data, optimize parameters so that the algorithms simulate human learning and thus learn to solve problem for data that the algorithms have not encountered before. ML algorithms provide deeper knowledge or predictions about the studied attribute or its evolution, some algorithms provide both. Good examples of machine learning implementations are an object recognition system that identifies objects learned by the system from the input image, or a decision support system that helps the physician diagnose a patient's illness and plan appropriate treatment for the situation.

Figure 2.1 below is given to provide comprehensive and intelligible understanding about generic flow of ML processing procedure.

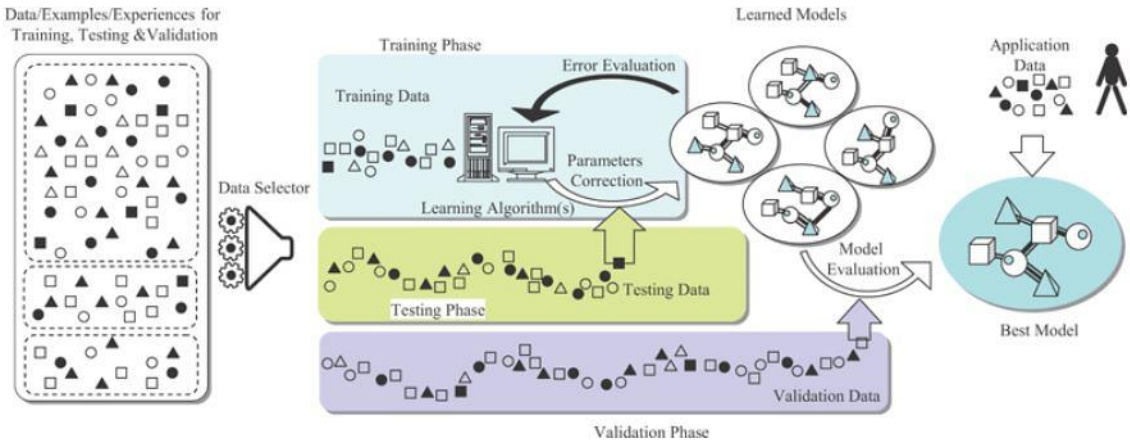

Figure 2.1 *Generic ML processing procedure flow [4].*

This process can roughly be divided into three parts as seen in Figure 2.1, the training phase, the testing phase, and the validation phase. First, the gathered data is processed to fit the requirements of the ML system. Depending on the ML system and the problem itself, data may be labeled or unlabeled. According to the criteria defined as suitable for

the problem, the data is divided into three independent and usually unconnected datasets. These datasets are called *training data*, *testing data* and *validation data*. In certain cases, depending on the ML system used, the validation set may not be needed and only training and testing data is enough for the process to succeed. Training data is fed into learning algorithms to generate models in the training phase. In the testing phase, the accuracies of the models generated are assessed against the testing data in terms of specific error functions. Model parameters are being tuned to improve the learning outcome at this point. This type of produce-and-modify cycle repeats as many times as needed for the quality standards of the model to be met or until the process is quit by the user. In the end of the process, either one or more models are generated, and the validation phase is then used to assess the validity of models. If the validity of models is insufficient, the learning process is restarted with new training data or a modification of the learning algorithm. Otherwise, model is approved and ready to be used in applications. [4]

2.2 Supervised learning

In supervised learning, set of pairs $\mathcal{D} = \{(\mathbf{x}_i, y_1), \dots, (\mathbf{x}_N, y_N)\}$ is given. This set is known as training set. Every instance $\mathbf{x}_i \in X \subseteq \mathbb{R}^d$ is an input vector of *d*-dimensions. For every instance $\mathbf{x}_i$ there is a corresponding categorial *label* $y_i \in Y = \{l_1, l_2, \dots, l_C\}$, where C is number of *class* categories. There is an unknown function $f : X \mapsto Y$ called *target function* which describes the relation between instances and their corresponding labels in training set. Goal of supervised learning methods is to find $\hat{f} \in \mathcal{H}$, which approximates target function f as accurately as possible. $\mathcal{H}$ is a set of hypotheses, set to reduce the number of possible functions needed to be considered in the process, making the process more manageable.[5]

Loss function measures the cost of false predictions and is defined as $\mathcal{L} : \mathcal{H} \times \mathcal{D} \mapsto \mathbb{R}^+$ [6]. The outcome of loss function describes, how well the found $\hat{f}$ fits to the training set, with value 0 indicating a perfect fit. To conclude, supervised learning problem can be presented as process to find $\hat{f} = \arg\min_{f \in \mathcal{H}} \mathcal{L}(f, \mathcal{D})$ [5].

2.3 Decision trees and tree ensembles

Decision trees are effective and accurate, yet conceptually and structure-wise simple, supervised learning algorithms. Data samples fed into a DT, are classified based on the results to the sequence of questions about their attributes related to the samples [7]. As in every tree structure, DT has *root node*, the topmost node in DT which, in most cases,

is *non-leaf node*. In addition to this, DT has ordinary non-leaf nodes called *internal nodes* from which data sample is assigned to either the left or right child as in the case of root node. Unlike in the case of root node and internal nodes, *leaf node* does not assign data samples anywhere, but instead gives us the final classification result. [5] Below Figure 2.2 is given to provide clear visual representation of the basic structure of very simple DT.

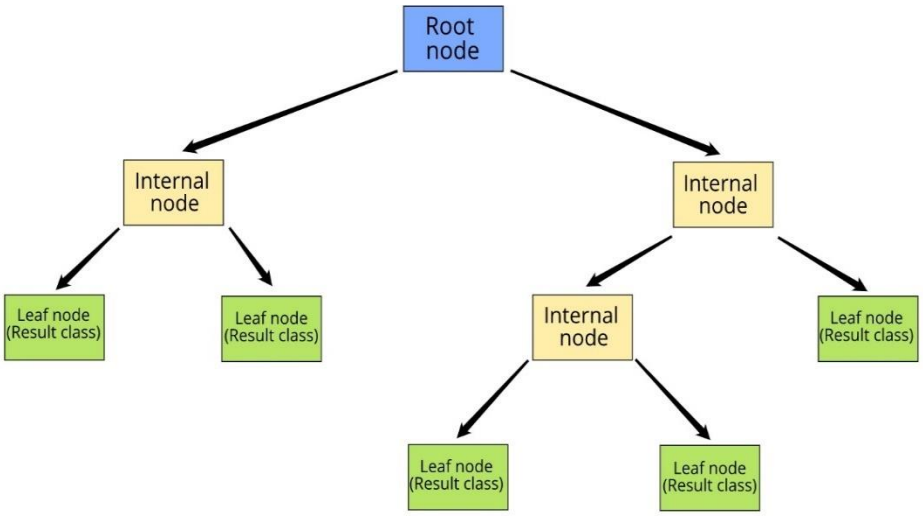

Figure 2.2 *Simple decision tree structure.*

First the data sample $\mathbf{x}_i$ is fed to the DT from the topmost node called the root. Root and other non-leaf nodes are represented as $\sigma(f, \eta, t_l, t_r)$, where t_l represents the left sub-tree whereas t_r represents the right sub-tree. Like in every non-leaf node, a comparison $\eta: \mathbf{x}_i^{(f)} \leq \eta$ for data sample is performed. In the comparison, f is the *splitting feature* and η is the *split threshold*. Based on the result of the comparison, the data sample is assigned into either left or right sub-tree. In the sub-tree's first node, previous node's child node, another comparison is made, and this cycle goes on until the data sample is assigned to leaf node $\lambda(\hat{y})$. Leaf node gives us the result $c(\mathbf{x}_i) = \hat{y}$ of DT as prediction of the class of the data sample. DT training to get more accurate prediction is done by finding optimal leaf constants, splitting feature f and split threshold η for the split, so that the loss function is minimized. [5]

Even though a single DT is already optimized, it may still suffer from high variance. In this situation, DT prone to overfit [5]. Variance can be decreased, and accuracy increased by forming a *tree ensemble*. Tree ensemble is constructed by bringing more

DTs to the system and combining the results of them. Two primary strategies for joining DTs are *random forests* (RF) and *boosting* [7].

2.3.1 Random forests

RFs are built on a method called *bagging*, which is a technique to decrease variance of ML model. For classification systems, such as DTs, bagging is carried out by choosing the class most voted by trees in an ensemble of trees. [8] Each tree in the ensemble is given a sample of the original training set. Samples are formed from the original training set by resampling method called *bootstrapping*, a technique that draws random sample vectors of dataset in a way that every vector is sampled individually with the equal distribution for every tree in the tree ensemble [5][9]. In addition to the samples drawn from the training set in bagging method, to reduce variance in RFs also the features are sampled. This is done, so that the trees in the ensemble must take in account a diverse set of features from other trees while considering a split [5]. If feature sampling is left out and the training set has strong features in it, variance would still stay high, since majority of trees would prefer to use those strong features in their top-most splits [5]. [9]

2.3.2 Boosted trees

Boosting is not just one algorithm that generates DT ensembles. In fact, boosting refers to big group of different algorithms that generate ensembles in different ways [10]. In this thesis, we are not going to go through every different boosting algorithm but just introduce the typical boosting procedure.

To demonstrate typical boosting procedure, we assume that single DT works with any distribution of data it is provided and consider the two-class classification problem as our example, which means that we are attempting to categorize samples as positive or negative. Our ground-truth function is f, and the training samples in dataset X are chosen from D which is our distribution in this case. We also assume that the dataset X is divided into three parts: X_1, X_2, and X_3, each representing for $\frac{1}{3}$ of the distribution, and that a DT using a random guess as prediction has a classification error of 50% on this task. We seek to find an accurate classifier for the task, but we have just one DT, that only has accurate classifications in dataset X_1 and X_2 and incorrect classifications in X_3. We can now say that our classifier denoted by DT_1 has $\frac{1}{3}$ classification error. It is clear that DT_1 is undesirable. The purpose of boosting is really to fix the faults of the DT_1. [10]

To begin, we seek to generate a new distribution D' from original D that highlights the flaws in DT_1 and emphasizes on the samples in X_3. Then we use D' to train a classifier called DT_2. Now we assume that DT_2 is a classifier with valid classifications in dataset X_1 and X_3 but incorrect classifications in dataset X_2. By merging DT_1 and DT_2, the resulting classifier will then have accurate classifications in dataset X_1, but possibly still errors in datasets X_2 and X_3. We create again a new distribution D'' to highlight the errors of merged classifier, then train a classifier DT_3 from it. Now DT_3 has accurate classifications in datasets X_2 and X_3. Therefore, by merging DT_1, DT_2, and DT_3, we get theoretically perfect classifier, because at least two of the three classifiers in each dataset of X_1, X_2, and X_3 make accurate classifications. [10] There are many boosting algorithms that work in principle as explained before, such as AdaBoost [11]. Of course, usually the problem assigned for boosted tree is not as simple as the problem just provided.

2.4 Robust decision trees

Robust DTs refer to a group of tree algorithms which are robust against adversarial examples. Chapter 2.4.1 introduces distance functions, popular tools for measuring distances between images. In Chapter 2.4.2, generic robust DT training procedure is introduced.

2.4.1 Distance functions

At this point, it is worth mentioning that for measuring distances between images, we use distance functions called ℓ_P-norms. Common ℓ_P-norms used in many articles and papers related to the topic of this thesis are, for example ℓ_0, ℓ_1, ℓ_2 and ℓ_∞. The ℓ_0 measures the amount of pixels which are different between to images, ℓ_1 measures the sum of absolute values of the two images, ℓ_2 is the Euclidean distance between two images and ℓ_∞ measures the absolute value of the maximum values of two images. [12] Usually, for robust DTs' adversarial training, more than one distance function is needed, since Kantchelian et al. [13] showed that training the tree to be robust measuring with only one distance function, makes the resulting model more prone to attacks in other distance functions [5]. Also, attack can be expressed as ℓ_P attack, where the adversarial examples' feature perturbations can be as an ℓ_P distance with limit of distance ϵ from the original feature.

2.4.2 Generic robust decision tree learning procedure

Chen et al. [2] showed in their study in 2019, that tree-based models like DTs are vulnerable when it comes to adversarial examples. Motivated by this, various robust decision tree algorithms have been developed that can withstand the adversarial examples created by attacks, while still producing accurate predictions. In general, not all algorithms created for this purpose follow the same principle, however, the presentation of the generic training procedure helps to understand the objectives of the operation of these algorithms, to see what variables the algorithms consider and what steps the adversarial training consists of.

Let us have a training set $\mathcal{D} = \{\mathbf{x}_i, y_i\}$ size of N samples. The training set has d real valued features so $1 \leq i \leq N,\ y_i \in \mathbb{R}$ and $\mathbf{x}_i = [x_i^{(1)},\ x_i^{(2)}, \dots, x_i^{(j)}, \dots, x_i^{(d)}] \in \mathbb{R}^d$. For simplicity, feature values in training set are normalized to $[0,1]$ so that $\mathbf{x}_i \in [0,1]^d$. At a given node, for generic DT model we can mark that the set of samples in the given node is denoted as $\mathcal{L} \subseteq \mathcal{D}$. From this point on in this chapter, we assume that the split happens with threshold η on the j-th feature. [2]

When it comes to traditional DT models, quality of each split in nodes can be evaluated with *score function* which is denoted as $S(\cdot)$. In other words, score function measures the quality of function that causes the split to either left child node $\mathcal{L}_L$ or right child node $\mathcal{L}_R$. Now we can denote that $S(\mathcal{L}_L, \mathcal{L}_R)$ and since the split at j-th feature is defined by the parameters in the function that causes the split, we can further denote that $S(j, \eta, \mathcal{L})$, where η is the threshold and $\mathcal{L}$ is the set of samples in the node. [2]

There are different scores to consider when it comes to choosing appropriate score function to discover the best split. In traditional DT models, score function could be for example Gini impurity or information gain [2]. A standard decision tree training procedure will assess the score function for all features and split thresholds possible on the leaf to be split, then choose the best j, η pair denoted as j^*, η^*. This procedure can be denoted as $j^*, \eta^* = \arg\max_{j,\eta} S(\mathcal{L}_L, \mathcal{L}_R) = \arg\max_{j,\eta} S(j, \eta, \mathcal{L})$. [2]

Since adversaries caused by attack can perturb, or in other words, modify features of our samples in $\mathcal{L}_L$ and $\mathcal{L}_R$ by any value inside ℓ_∞ ball of radius ϵ, now for each sample $\mathbf{x}_i$: $B_\epsilon^\infty(\mathbf{x}_i) := \left[x_i^{(1)} - \epsilon, x_i^{(1)} + \epsilon\right] \times \dots \times \left[x_i^{(d)} - \epsilon, x_i^{(d)} + \epsilon\right]$ [2]. Now that our samples' features can be perturbed by any value inside ℓ_∞ ball of radius ϵ, these perturbations attempt to minimize our score function so that our model would perform as inaccurately as possible.

For robust DTs, the procedure for finding the optimal j, η pair is usually different from traditional DT models. Of course, the result obtained by model affected with kind of attacks described above would not be desirable, thus our procedure needs to be changed to $j^*, \eta^* = \arg\max_{j,\eta} RS(j, \eta, \mathcal{L})$, where RS stands for *robust score function*. RS is denoted as $RS(j, \eta, \mathcal{L}) \coloneqq \min_{\mathcal{L}'=\{(\mathbf{x}_i', y_i)\}} S(j, \eta, \mathcal{L}')$, where $\mathbf{x}_i' \in B_\epsilon^\infty(\mathbf{x}_i)$ and $\mathbf{x}_i' \in \mathcal{L}'$. The $\mathcal{L}'$ is set of samples from the training set, which are now perturbed by adversaries so that this set would lead to worst possible score in split for the j-th feature with threshold η . $\mathbf{x}_i'$ presents one sample in $\mathcal{L}'$, which is perturbed. Now it can be said that our current procedure $j^*, \eta^* = \arg\max_{j,\eta} RS(j, \eta, \mathcal{L})$ has become max-min optimization task, our goal being: given all possible perturbations inside ℓ_∞ ball of radius ϵ, maximize the minimum score achieved. Computationally, this calculation becomes very heavy and inefficient, because at a single node for single split we would have to calculate the max-min optimization calculation for each feature in every example in the node. [2]

At this point, *ambiguity set* $\Delta\mathcal{L}$ is introduced. Every sample in the training set, which attacker is able to change from left split to right split and vice versa, is in ambiguity set. Those samples, that can not be changed from other split to another by any action that attacker does, are in the *non-ambiguity set.* [2] Ambiguity sets and non-ambiguity sets are defined for the left and right split separately, and more detailed representation for those sets are shown below. Representation is based on the presentations in [2] and [5].

$\Delta\mathcal{L} \coloneqq \mathcal{L} \cap \{(\mathbf{x}_i, y_i) \mid \eta - \epsilon \leq x^{(j)} \leq \eta + \epsilon\}$	ambiguity set
$\Delta\mathcal{L}_R \coloneqq \Delta\mathcal{L} \cap \mathcal{L}_R$	right ambiguity set
$\Delta\mathcal{L}_L \coloneqq \Delta\mathcal{L} \cap \mathcal{L}_L$	left ambiguity set
$\Delta\mathcal{L}_R^o \coloneqq \mathcal{L}_R \setminus \Delta\mathcal{L}$	right non-ambiguity set
$\Delta\mathcal{L}_L^o \coloneqq \mathcal{L}_L \setminus \Delta\mathcal{L}$	left non-ambiguity set

With above sets defined, we can consider four different cases, that represent attacker behavior: (1) there are not any perturbations: $S_1 = \mathrm{S}(\mathcal{L}_R, \mathcal{L}_L)$; (2) all samples in $\Delta\mathcal{L}$ are perturbed to right $S_2 = \mathrm{S}(\mathcal{L}_R^o, \mathcal{L}_L^o \cup \Delta\mathcal{L})$; (3) all samples in $\Delta\mathcal{L}$ are perturbed to left $S_3 = \mathrm{S}(\mathcal{L}_L^o \cup \mathcal{L}_R^o, \Delta\mathcal{L})$; samples in $\Delta\mathcal{L}$ are swapped $S_4 = \mathrm{S}(\mathcal{L}_L^o \cup \Delta\mathcal{L}_R, \mathcal{L}_R^o \cup \Delta\mathcal{L}_L)$. In all the four cases, S before the notations still refers to score function. Now we calculate minimum of above defined four cases and we obtain computationally light and efficient approximation for RS so that $RS(j, \eta, \mathcal{L}) \approx \min\{S_1, S_2, S_3, S_4\}$. This rough approximation operates on $O(1)$ time complexity. [2] As this is the general robust DT learning procedure

to find optimal splits, leading to more accurate models, it is worth mentioning that all the other robust DT learning algorithms, besides robust trees implemented by Chen et al. [2], vary more or less from the above explanation.

3. METHODOLOGY

To give understanding about the performed experiment in Chapter 4, this section discusses attacks, attack models, DT algorithms, and evaluation methods used in experiment in more detail. Experiment carried out in this thesis is based on the experiment done in article of Vos et al. [14] conducted in 2021. Codes for the experiment are found from the GitHub [15] of the original paper. Chapter 3.1 goes through basic nature of attacks in the context of adversarial attacks. In Chapter 3.2, models used in the experiment are introduced and the Chapter 3.3 focuses on evaluation part of the experiment.

3.1 Attacks

All the attacks we discuss and are used in the experiment, are adversarial attacks, in which attacker uses adversarial examples. Adversarial examples are carefully perturbed data samples, that fool the DT model to give false predictions [3]. Below Figure 3.1.1 has been provided to give understanding how these already harmful, but minor modifications seem to human eye. If these kinds of examples seen in the Figure 3.1.1, get into the procedure of training, testing, or validating the DT model, resulting DT model might make inaccurate predictions. Needless to say, inaccurate predictions in DT model, working in for example medicine related tasks, might lead to unfortunate outcomes for the patient or the subject of the study.

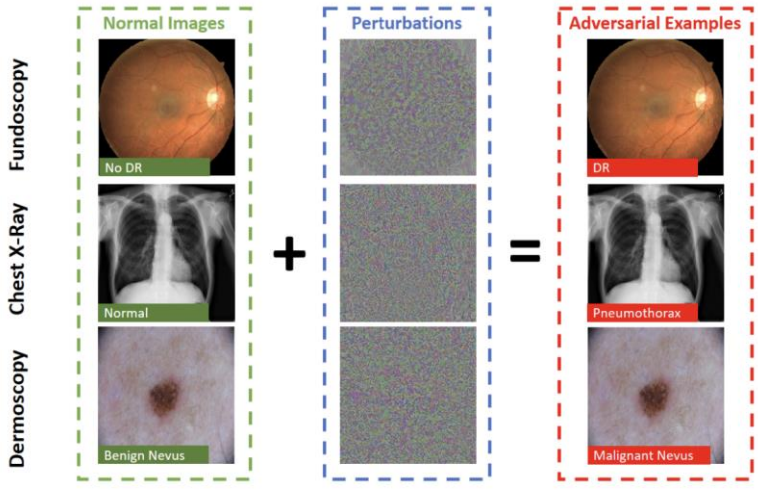

Figure 3.1 *Examples of adversarial examples in the field of medicine [16].*

In case of robust DT models, it is not always simple for the attacker to be successful. Some of the robust DT algorithms are so robust, that the adversarial examples must have big and, in some cases, very noticeable perturbations to fool the model. Different attack algorithms have been developed for making the adversarial examples, these algorithms are not discussed here more specifically. In general, attacks where attacker has unlimited access to parameters of the DT, internal functions, even some cases training data, are called *white-box attacks. Black-box attacks* on the other hand, are attacks, where attacker is able to only feed adversarial examples to the DT [17].

In the experiment of our interest, focus is set to create many different attack types, so that performance of robust DTs could be assessed more comprehensively [14]. Chapter 3.1.1 describes, how the attacks can be modified, and what kind of attack algorithm is used in the experiment.

3.1.1 Attacker model

In the experiment, it is assumed that there is an attacker who knows the model, thus is able to utilize an attack model, specified by the user while perturbing the examples. Limits for perturbations for each separate feature are set by the user. [14]

Since user can set the limits for perturbations, different possible options for setting the limits for a feature are listed below. Representation of the possible settings below is inspired by presentation in [14].

- Feature can be set in a way, that it cannot be perturbed.
- Feature can be set in a way, that it decreased or increased.
- Feature can be set in a way, that it can be perturbed only to a lower value.
- Feature can be set in a way, that it can be perturbed only to a higher value.
- Feature can be set in a way, that it can be perturbed to any given value.
- Feature can be set in a way, that it can be perturbed by distance of ϵ. If this setting is done for all the features, the attack with this perturbation limit can be expressed as ℓ_∞ attack with radius of ϵ.
- Feature can be set in a way, that it can be perturbed either to the right by amount of ϵ_r or to the left by amount of ϵ_l.

In the experiment, desired attack models are created with the help of MILP (Mixed Integer Linear Program [13]). MILP in the experiment, is slightly modified from the original MILP, so that the run time could be improved [14]. These attacks can be referred as MILP attacks and they were introduced by Kantchelian et al. in [13]. MILP is and algorithm, that computes the smallest perturbation for given distance function (ℓ_0, ℓ_1, ℓ_2 or ℓ_∞), that causes the model to misclassify the sample [13].

3.2 Models

In Chapters 3.2.1 and 3.2.2, different DT and DT ensemble models used in the experiment are briefly introduced. 3.2.1 goes through non-robust and robust DT models and Chapter 3.2.2 goes through non-robust and robust DT ensemble models used in the experiment. No multi-class classification is done in the experiment. Every model used in the experiment, is working on binary classification problems, even though some of the models would support multi-class classification. Models that are introduced in Chapters 3.2.1 and 3.2.2 are compared to each other in the experiment. In Chapters 3.2.1 and 3.2.2, used models' parameters, input data and limitations are not discussed. Data used in the experiment, limitations set by user and other input for the used models are discussed in the Chapter 4 and its sub-chapters.

3.2.1 Tree models

Probably the simplest model used in the experiment is standard non-robust DT. Standard DT classifier in the experiment is implemented using Scikit-learn's [18] `tree` library's `DecisionTreeClassifier` -function [15]. Standard DT uses Gini impurity to score the splits [18].

Chen et al. tree is a robust DT model, originally introduced and implemented by Chen et al. in [2]. In the source code of the experiment, Chen et al. tree is implemented with source code's own `GrootTreeClassifier` -function, but the implementation is based on the implementation of Chet et al. in [2]. Chen et al. tree uses worst case Gini Impurity, introduced in [2] to score the splits [15].

TREANT tree is robust DT model. TREANT was introduced by Calzavara et al. [6] in 2020. In the source code of the experiment, TREANT model is almost directly copied from the GitHub of the original implementation of Calzavara et al. in [19]. Small adjustments were made to the experiments source code by Vos et al. so that the ℓ_∞ could be encoded properly [15]. TREANT does not use splitting criterion and it optimizes

the loss function directly which leads to long running times of the algorithm since it needs to pre-compute every potential attack and the loss function is minimized in every step of the building stage of the tree [6][14].

GROOT tree is robust DT model, introduced by Vos and Verwer [14] in 2021. When it comes to learning procedure of the GROOT tree, it is very similar to traditional DT algorithm which is discussed in Chapter 2.3 of this thesis. Unlike traditional DTs, GROOT uses a robust score function and it explores through more possible splits than the traditional DT. On top of that, GROOT propagates samples according to the attacker in the process, to account for data samples that the adversarial attack might move. [14] In the case of traditional DT models, propagation of the samples is not done. GROOT tree uses adversarial Gini Impurity to score the splits [14]. Algorithm of the GROOT tree is implemented in its entirety in the source code of the experiment in [15] and it is more thoroughly discussed in [14].

3.2.2 Ensemble models

Gradient boosting is a non-robust method for making tree ensembles. Gradient boosting falls under the concept of boosted trees, which is conceptually and from the point of basic function discussed in Chapter 2.3.2 of this thesis. Simple gradient boosting used in the experiment is implemented using Scikit-learn's [18] `ensemble._gb` library's `GradientBoostingClassifier` -function.

Traditional non-robust random forest used in the experiment is implemented using Scikit-learn's [18] `ensemble._forest` library's `RandomForestClassifier` -function [15]. Basic function and idea behind random forest is discussed in Chapter 2.3.1 of this thesis.

In the original study, in which this experiment is based on, Chen et al. boosting was part of the accuracy, efficiency and robustness testing. It was also compared to other robust and non-robust ensemble and tree models [14]. Chen et al. boosting, is a boosting method, first introduced in [2] by Chen et al. In this experiment, Chen et al. boosting is left out from the experiment due to troubles encountered while trying to train the model.

Chen et al. forest is a robust DT ensemble model, originally implemented by Chen et al. in [2]. In the experiment, Chen et al. forest is implemented with experiment's source code's own function `GrootRandomForestClassifier`, but the implementation results to similar model as seen in [2]. [15] Chen et al. forest is forest ensemble of Chen et al. trees [2]. DT random forests and their basic function is discussed in the Chapter 2.3.1 of this thesis.

Provably robust boosting is a robust boosted tree method implemented by Andriushchenko et al. in [20] in 2019. In provably robust boosting, by using boosted decision stumps, upper bound for the adversarial loss is computed and optimized for ensemble of trees to obtain accurate ensemble of shallow trees [14]. Decision stumps are trees, which have depth of one [20]. In the experiments source code, code for the provably robust boosting is almost directly copied from the GitHub of the original provably robust boosting implemented in [20] by Andriushchenko et al. Only minor changes have been made. [15]

GROOT forest is a robust DT ensemble model implemented by Vos and Verwer [14]. GROOT forest is fully implemented in the source code of the experiment in [15]. Idea of random forests is introduced in Chapter 2.3.1 of this thesis. GROOT forest is an ensemble of GROOT trees, which gives prediction by first averaging and later rounding all the predictions given by different GROOT trees in the ensemble [14].

In the Table 3.1 models used in the experiment of this thesis are presented. The year in which the model was published, and information whether the model is designed or is not designed to be robust is shown for every model. First, the tree models are listed and below them are the ensemble models.

Table 3.1 *Models used in the experiment, their year of publication and designed state of robustness.*

Model	Year	Robust/Non-robust
Decision Tree (Scikit-learn)	2011	Non-robust
Chen et al. Tree	2019	Robust
TREANT Tree	2020	Robust
GROOT Tree	2021	Robust
Gradient Boosting (Scikit-learn)	2011	Non-robust
Random Forest (Scikit-learn)	2011	Non-robust
Chen et al. Forest	2019	Robust
Provably Robust Boosting	2019	Robust
GROOT Forest	2021	Robust

3.3 Evaluation

In chapter that discusses evaluation, methods for evaluating the parameters of interest, efficiency, accuracy and robustness of the models, are introduced. Evaluation is crucial in machine learning, in order to find the most suitable models for the task. Evaluation can be done for large variety of metrics and from many different points of views, still in this thesis evaluation focuses on how the build models perform when it comes to efficiency,

accuracy and robustness. Evaluating and measuring efficiency is covered in Chapter 3.3.1, accuracy and robustness are discussed in Chapter 3.3.2.

3.3.1 Efficiency

Measuring the efficiency of the algorithms used in this thesis is relatively simple. The only parameter used to measure efficiency is run time [14]. Run time is the time during which the algorithm performs its actions. Run time is measured by measuring the time that it takes for the algorithm to finish from the starting of the algorithm. Run times of the algorithms would not tell us much if they were not measured in comparison to other algorithms performing the same task. In this context, it is more significant to know how fast the algorithms perform compared to other similar algorithms than the fact that we only measure the run times of individual algorithms.

3.3.2 Accuracy and robustness

The accuracy of the models for structured datasets in this thesis is measured by few different accuracy parameters. The first parameter is *mean accuracy*. Mean accuracy is the average accuracy of the model over all 14 structured datasets used for the model [14]. For example, if the accuracy of the model for dataset 1 is 0.90 and for the dataset 2 accuracy is 0.80, models mean accuracy is calculated as follows: $\text{Mean accuracy} = \frac{(0.80+0.90)}{2} = \frac{1.70}{2} = 0.85$ which equals to 85%. For each structured dataset, accuracy is measured by comparing the classifications of the models to the ground-truths of the dataset's samples. The percentage of data samples that the model correctly predicted for one dataset, is the accuracy of the model for that dataset. At this point, the dataset does **not** yet include adversarial examples. This allows the accuracy of the models to be compared with each other in the absence of an attack.

Another parameter that measures accuracy of the model for structured datasets in this thesis is *mean adversarial accuracy* [14]. Mean adversarial accuracy is calculated in the same way as mean accuracy, but this time the dataset includes perturbed samples with perturbations inside ℓ_∞ ball of radius ϵ [14]. One can now compare how an individual model works in terms of accuracy without attack versus during attack. One can also compare the accuracies of different models under attack.

In the article by Vos et al. [14], which in this experiment is based, models are also ranked by their number of wins over 14 structured datasets. Ranking method used is mean rank [14]. Mean rank is calculated differently to standard ranking where best model would be

ranked as 1., the second best as 2. and so on. In mean ranking, rank of the model is average of all of the models ranks. Similarly to standard ranking, mean rank can be read as: smaller the mean rank, the better the model performed in the comparison of the models. Model with the lowest mean rank is the best model, model with the highest mean rank is the worst model. Mean rank is not playing an important role in the experiment done in this thesis, but it is used in the [14], in which this thesis is closely related. This is why the mean rank is at least worth mentioning.

Robustness of the model for image datasets is assessed by calculating the average perturbation distance necessary to induce a misclassification [14]. This metric tells us, how much should the data sample be perturbed, so that it would fool the model and force misclassification. Higher the necessary perturbation distance, better the model since attacker must make bigger changes into the data samples in order to fool the model. In some cases where the model is robust enough, perturbations needed to trick the model must be so high, that the perturbed samples can be easily recognized by the human eye.

4. EXPERIMENT

In the experiment of this thesis, goal is to produce results that answer the research questions of this thesis. In the experiment, comparison of various state-of-the-art robust DT models and other non-robust DTs is done. Focus of comparison is set to the following attributes of the models: efficiency, accuracy, and robustness. Codes run in the experiment produce and provide results on all attributes of the models mentioned before, and with help of those results, research questions are answered.

Experiment done in this thesis, is a reproduction of the experiment conducted by Vos et al. in [14]. Source code for the experiment is retrieved from GitHub [15]. Codes are implemented by Vos et al. [15]. Bu running the source codes, original experiment done in paper by Vos et al. [14] can be reproduced.

In Section 4.1, data used in the experiment is reviewed. Section 4.2 introduces the part of experiment where models are trained, and Section 4.3 explains the result producing part of the experiment. Section 4.4 goes through parameter limitations and introduces the hardware used while running the experiment codes. In Section 4.5 results of the experiment are reviewed and discussed.

4.1 Data

In the experiment, two different kind of data is used. For all the models, 14 different structured datasets are used. Two different image datasets are used only for random forest, Chen et al. forest, GROOT forest and Provably robust boosting.

Structured datasets were retrieved from OpenML introduced in [21]. In the Table 4.1 is the summary of the used structured datasets. On the first column the name of the dataset is given, on the second column is the number of samples in the dataset, on the third column is the number of features in the dataset's samples and on the last column the radius ϵ of the ℓ_∞ ball of possible perturbations to the dataset in the case of attack is given. The information given in the Table 4.1 is produced by script `summarize_datasets.py` in the source code of the experiment. Style of the Table 4.1 is inspired by similar table in [14].

Table 4.1 *Summary of the structured datasets used in the experiment.*

Dataset (OpenML version)	Samples	Features	ϵ
haberman	306	3	0.1
blood-transfusion	748	4	0.1
planning-relax	182	12	0.1
cylinder-bands	277	37	0.1
SPECTF	267	44	0.1
diabetes	768	8	0.05
parkinsons	195	22	0.1
ionosphere	351	34	0.2
sonar	208	60	0.1
climate-model-simulation	540	18	0.1
banknote-authentication	1372	4	0.1
breast-cancer	683	9	0.3
wine	6497	11	0.05
spambase	4601	57	0.05

In the experiment, two different image datasets were used. These datasets were MNIST 2 vs 6 and Fashion-MNIST sandals vs sneakers [15]. Like in the case of structured datasets, also MNIST 2 vs 6 and Fashion-MNIST sandals vs sneakers -datasets were retrieved from the OpenML [21].

4.2 Training the models

Script `train_kfold_models.py` in the source code of the experiment performs the training of all the models used in the experiment. In this script, training phase parameters for the tree models are set [15]. All structured datasets are used in the training phase of the models. Parameter limitations are discussed further in Section 4.4 of this thesis. Script `train_kfold_models.py` measures for each model, how much time did the training of the model take. Script also outputs the training times as an csv file, so that the training times can be assessed further. [15] Script writes the trained models in to the correct subdirectories in the correct format. In the subdirectories, codes can be retrieved by next script that generates other results of the experiment.

4.3 Generating the results

Two different scripts are responsible for visual outputs of the source code. Script `generate_kfold_results.py` generates output images for results on accuracy and efficiency. Script `generate_kfold_results.py` retrieves the trained models from the subdirectories, performs the MILP attack by Kantchelian et al. [13] for all the trained

models and outputs result images [15]. Result images hold information about models' runtimes and accuracies.

Script `image_experiments.py` trains and evaluates the tree ensemble models with image datasets. Script also provides visual outputs containing information about optimal adversarial examples.[15] With optimal adversarial examples and their corresponding ℓ_∞ distances, models' robustness can be assessed.

4.4 Parameter limitations and hardware

Since components of the computer can influence the runtimes and outcome of the codes, the relevant information about the components of the computer is provided in this chapter. The codes in this experiment were run with computer that had AMD Ryzen 7 PRO 5850U processor with 8 CPU cores and 8 GPU cores. Computer had 32 GB of RAM.

Parameter and training limitations for the models were set in script `train_kfold_models.py`. Each model is trained with 5-fold stratified cross validation for each structured dataset. To preserve interpretability, all individual DTs were trained to a depth of four. To create a split, all models needed at least ten data samples, and to make a leaf, at least five data samples. On the same feature, it was permitted for all the models to split numerous times. The total number of trees trained in each ensemble was restricted to 100. [14]

For the MNIST dataset, `image_experiments.py` was run with epsilon (ϵ) value of 0.4 and for the Fashion-MNIST the epsilon was also 0.4. Epsilon (ϵ) is the assumed perturbation radius during training [15]. `image_experiments.py` was set to train the models from scratch and 100 trees were used in every ensemble.

4.5 Results

Running of the necessary source codes from [15] in PyCharm environment, resulted to several outputs. In Chapter 4.5.1 outputs that contain information about efficiency of the generated models are discussed. Chapter 4.5.2 goes through results related to accuracy and Chapter 4.5.3 discussed the robustness of resulted models. Result images and overall results are discussed more in the Chapter 5.1, where final conclusions for the experiment are given and research questions are answered.

4.5.1 Results on efficiency

For efficiency, source code's script `generate_kfold_results.py` in [15] generated us result figure, Figure 4.1, from the runtimes of the algorithms that generated the models. In the Figure 4.1, on the x-axis, used dataset is seen, and on the y-axis, we can see the runtime in seconds. In the upper image of the Figure 4.1, runtimes are plotted for the tree models and in the lower image, runtimes for ensemble models can be seen. Note that the x-axis is shared between two images seen in the Figure 4.1, so the dataset names can be seen only in the lower image. In the upper image, bars are plotted for the same dataset as in the lower image.

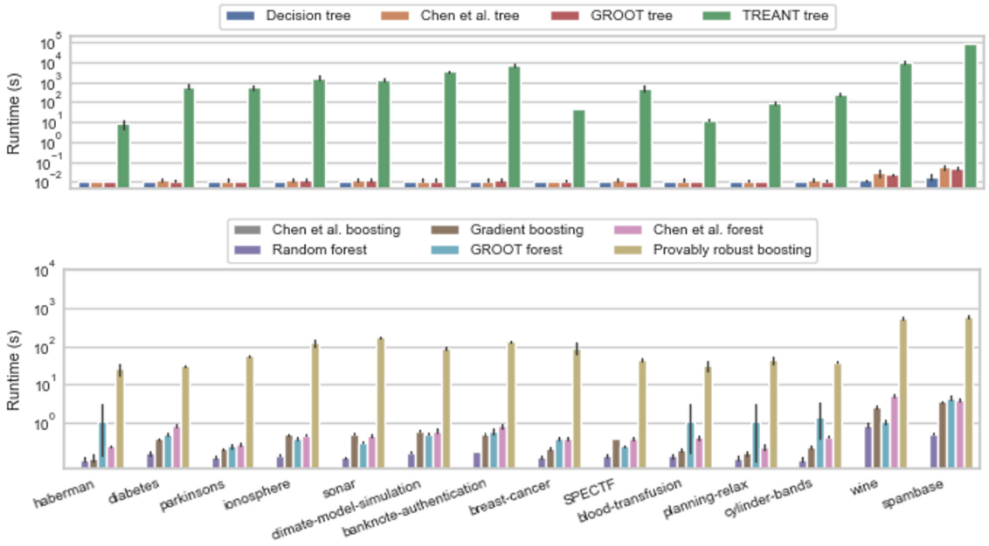

Figure 4.1 *Runtimes of the algorithms.*

As can be seen from the results for runtimes in the Figure 4.1, slowest algorithm to run for all the datasets from the tree models in experiment is TREANT tree, which ran over a day. Slowest algorithms of the ensemble models in the experiment to run for all the datasets was provably robust boosting, which ran several hours. GROOT tree and Chen et al. tree seems to perform on the same level as standard DT when it comes to runtime, they all ran in the matter of fraction of a second. GROOT forest, Chen et al. forest and gradient boosting perform somewhat similarly when it comes to runtimes. Random forest seems to be the fastest ensemble model to run for all the datasets used.

4.5.2 Results on accuracy

For accuracy, source code's script `generate_kfold_results.py` in [15] generated us result figures, Figure 4.2 and Figure 4.3. In Figure 4.2, mean accuracy of the tree and

ensemble models used in the experiment are seen. In Figure 4.3, mean adversarial accuracy for the tree and ensemble models used in the experiment are seen. In both Figure 4.2 and Figure 4.3, on the x-axis, there is the name of the model. On the y-axis in Figure 4.2, there is the mean accuracy of the model over all the used structured datasets and on the y-axis of Figure 4.3, there is the mean adversarial accuracy of the model over all the used structured datasets. Note that there are no adversarial examples involved in the datasets, that are used for generating the results in Figure 4.2. In the other hand, there are adversarial examples in the datasets that leads to the seen results in Figure 4.3.

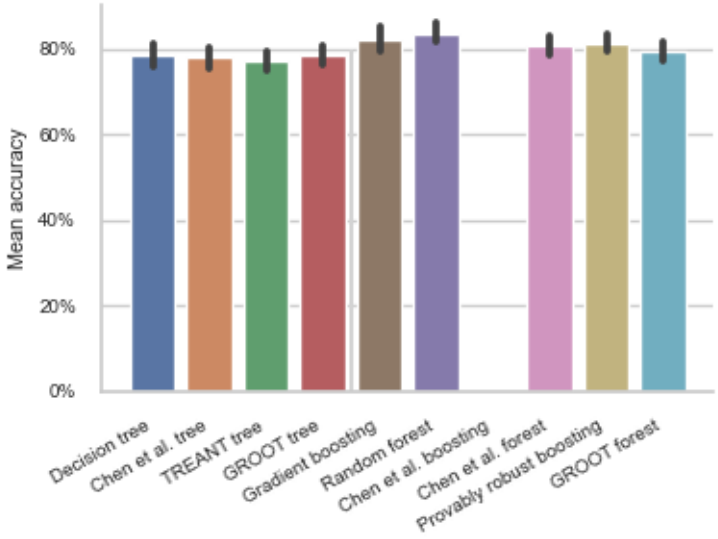

Figure 4.2 *Mean accuracies of the generated models.*

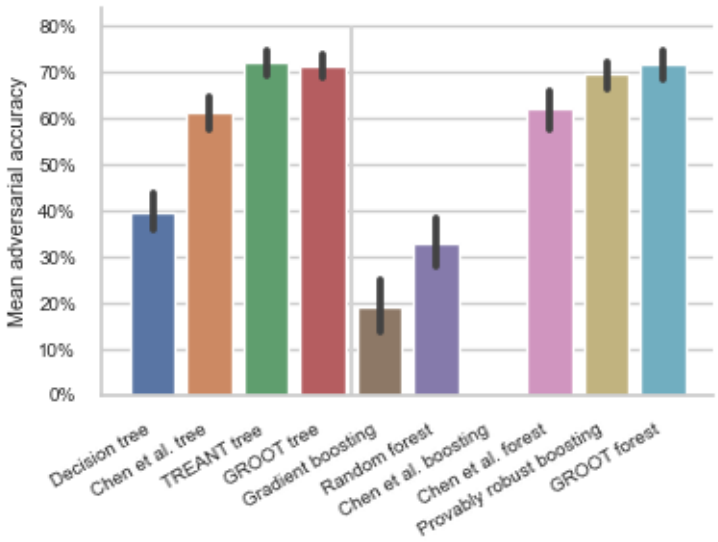

Figure 4.3 *Mean adversarial accuracies of the generated models.*

As can be seen from the mean accuracies of the models in Figure 4.2, all the models performed relatively well and very similarly in the case where there were not any perturbed samples in the datasets. Mean accuracies of all the models where around 80%. When it comes to mean adversarial accuracies, there is a lot of differences between the models' performances. For all the models, accuracy has decreased when compared to the case where there were no adversarial examples involved. From the tree models, TREANT and GROOT tree performed the best, both scoring a mean adversarial accuracy around 70%. Their accuracy dropped around 7-8% from the case where there were not adversarial examples. From the ensemble models, provably robust boosting and GROOT forest performed the best, both scoring a mean adversarial accuracy around 70%, so they had quite similar drops and accuracies to GROOT tree and TREANT. Chen et al. tree and Chen et al. forest both dropped just a bit under 20% from mean accuracy when adversarial examples were added to the datasets, both scoring a mean adversarial accuracy around 60%.

In the case of adversarial examples in the datasets, non-robust DT and DT ensemble models all lost plenty of accuracy from the case where there were not any adversarial examples in the datasets. Standard DT scored a mean adversarial accuracy of just under 40%, so it lost just under 40% of accuracy compared to the case where there were not any adversarial examples. Random forest scored a mean adversarial accuracy of just over 30%, dropping 50% of mean accuracy compared to the case without adversarial exampled in datasets. Gradient boosting had the biggest drop in mean accuracy of all the models. Scoring mean adversarial accuracy of just under 20%, gradient boosting dropped over 60% of its mean accuracy compared to the case where the datasets are free from perturbed samples. In the Table 4.2 models used and their changes in accuracies in the comparison of mean adversarial accuracy and mean accuracy has been summarized.

Table 4.2 *Models used in the experiment, their approximated change of accuracy (mean adversarial accuracy versus mean accuracy) and their designed state of robustness.*

Model	Approx. change of accuracy (Mean adversarial accuracy vs. mean accuracy)	Robust/Non-robust
Decision Tree (Scikit-learn)	$-40\ \%$	Non-robust
Chen et al. Tree	$-20\ \%$	Robust
TREANT Tree	$-7\ \%$	Robust
GROOT Tree	$-8\ \%$	Robust
Gradient Boosting (Scikit-learn)	$-60\ \%$	Non-robust
Random Forest (Scikit-learn)	$-50\ \%$	Non-robust
Chen et al. Forest	$-20\ \%$	Robust
Provably Robust Boosting	$-11\ \%$	Robust
GROOT Forest	$-8\ \%$	Robust

4.5.3 Results on robustness

For robustness, source code's script `image_experiments.py` in [15] generated us result figures Figure 4.4 and Figure 4.5. In the Figure 4.4 and Figure 4.5, are seen the minimal adversarial examples. Minimal adversarial examples are the least perturbed examples needed to fool the model. Above the images are the actual ℓ_∞ distances of the minimal adversarial examples to the original images [14]. Larger the ℓ_∞ distance, better the model. This is since larger ℓ_∞ distance means that the attacker must perturb the original sample more to fool the model. Dataset used for the `image_experiments.py` to generate the results in the Figure 4.4 was MNIST 2 vs 6 and Fashion-MNIST sandals vs sneakers in the Figure 4.5. In both Figure 4.4 and Figure 4.5, on the x-axis above the images are seen, which DT ensemble model's result images are seen on the exact column below. For the MNIST dataset, `image_experiments.py` was run with epsilon (ϵ) value of 0.4 and for the Fashion-MNIST the epsilon was also 0.4. All the ensemble methods except gradient boosting were included to the runs of `image_experiments.py`. Robustness was evaluated for the image datasets, without counting runtimes. Efficiency of the models is only evaluated for the structured datasets back in Section 4.5.1 of this thesis.

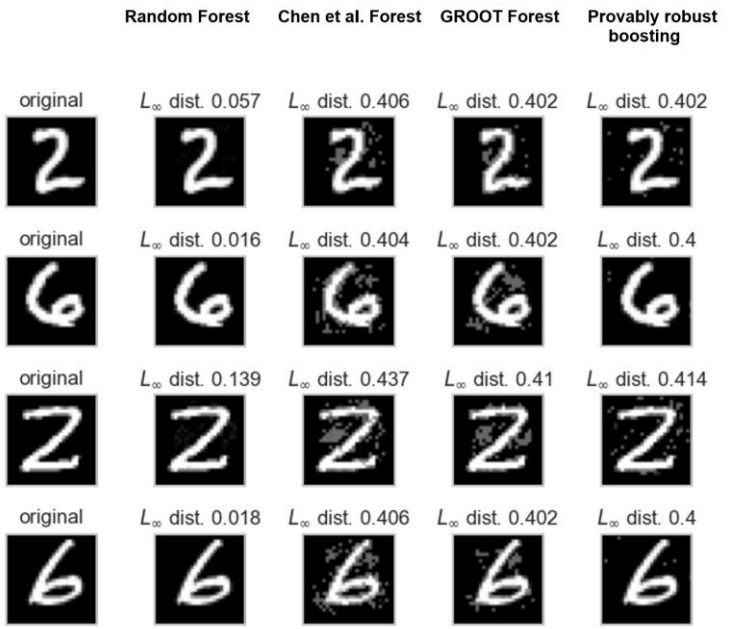

Figure 4.4 *Result on robustness with MNIST 2 vs 6.*

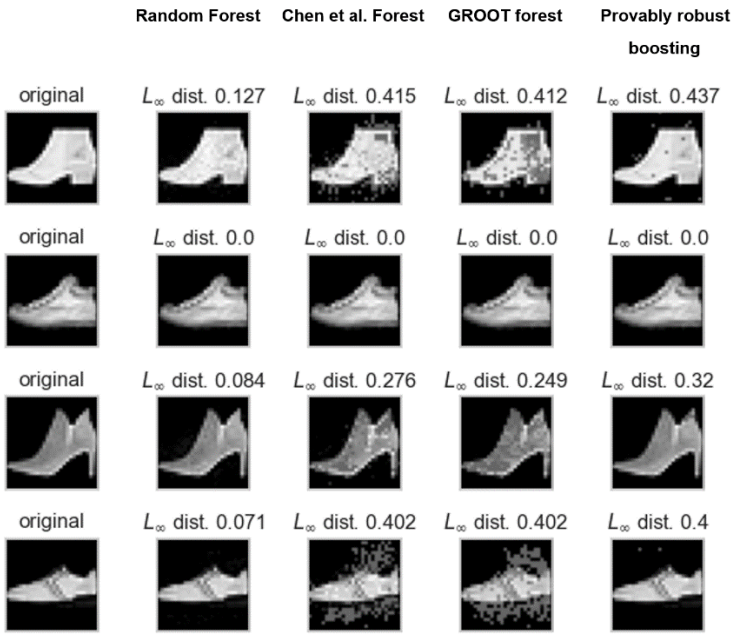

Figure 4.5 *Result on robustness with Fashion-MNIST sandals vs sneakers.*

First thing that can be noticed from the Figure 4.5, is that the second image row for all the models scored minimum ℓ_∞ needed for misclassification as 0.0. This is due some error in the process of code `image_experiments.py`. Second option is that every model misclassified the image without any perturbations made to the image.

As we can see from the Figure 4.4 and Figure 4.5, the only non-robust ensemble model included in the robustness analysis, random forest, did not perform as well as the other models. This is the case for all the images in both figures, besides the second image in the Figure 4.5, where the image was misclassified by all the models without any perturbations.

Minimum ℓ_∞ needed for misclassification is very similar for all the robust DT ensemble models in Figure 4.4. There are no significant differences between results on robustness for the robust models in any of the images in Figure 4.4. Biggest difference on same image in Figure 4.4 in terms of minimum ℓ_∞ is on the third image, where difference in ℓ_∞ values for Chen et al. forest and GROOT forest is 0.027 in favor of Chen et al. forest. Again, when it comes to the results in Figure 4.5, there are any huge differences between robust models' performances for the same image. Biggest difference for robust models on same image in Figure 4.5 in terms of minimum ℓ_∞ is on the third image, where difference in ℓ_∞ values for GROOT forest and Provably robust boosting is 0.071 in favor of Provably robust boosting.

5. CONCLUSIONS

DTs are powerful machine learning algorithms, which are impossible to implement in precision and security sensitive everyday applications without them withstanding adversarial attacks. This thesis was done to compare different state-of-the-art robust DTs to each other and to other non-robust DTs in terms of accuracy, efficiency, and robustness.

In this thesis, it was seen that all the robust DTs and tree ensembles were at least 20% more accurate in prediction under adversarial attacks than traditional DT and tree ensembles. Best robust DT mean accuracy-wise, TREANT tree performed approximately 30% better in accuracy under adversarial attack than normal DT. Best robust DT ensemble mean accuracy-wise, GROOT forest performed approximately 38% better in accuracy under adversarial attack than normal random forest, which was the best non-robust performer under adversarial attack.

Interestingly, GROOT tree, which achieved approximately 2% less mean adversarial accuracy than TREANT tree, was almost as fast as normal DT for majority of the structured datasets when it comes to runtime of the learning. GROOT tree's runtime was always under a second for all structured datasets used, while the runtime of TREANT tree was at its best approximately 10 seconds and in worst case over one day. From the robust ensembles, GROOT forest, which performed the best mean adversarial accuracy-wise from all the ensembles, was faster than other less accurate robust ensemble methods in 8 of the 14 structured datasets used. Chen et al. forest was faster or as fast as GROOT forest in 6 of the 14 structured datasets used, but it also scored approximately 10% less mean adversarial accuracy than GROOT forest. Results in the thesis show, that there was only little difference between robust Dt ensembles when it comes to robustness.

With different parameter limitations, the models might have been trained to become even more accurate under adversarial attacks. If the models might have become more accurate due to looser parameter limitations, also the runtimes might have become higher. Additionally, hardware may have affected the results at least in efficiency. By giving the experiment more computing power, the models could have been trained faster. Moreover, Chen et al. boosting would have been interesting to study, if I didn't run into some errors while trying to run the code.

In the future, it would be nice to see more accurate and fast robust DTs, that can in addition to binary classification, work on multiclass classification tasks. In conclusion, even though robust DT models are already a lot more accurate classifiers than normal DT models under adversarial attack, the models still need to improve in accuracy while performing time efficiently before they can be used reliably in real life everyday applications.

REFERENCES

[1] Cao N, Li G, Zhu P, Sun Q, Wang Y, Li J, et al. Handling the adversarial attacks: A machine learning's perspective. Journal of ambient intelligence and humanized computing. 2018;10:2929–2943. Available:https://link.springer.com/article/10.1007/s12652-018-0714-6

[2] Chen, H., Zhang, H., Boning, D. and Hsieh, C.-J. Robust Decision Trees Against Adversarial Examples. Proceedings of the 36th International Conference on Machine Learning. ICML 2019, 9-15 June 2019, Long Beach, California, USA. 2019. Available: arXiv:1902.10660 [cs.LG]

[3] Papernot N, McDaniel P, Goodfellow I. Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples. 24 May 2016. Available: arXiv:1605.07277 [cs.CR]

[4] Lai, C.-C., Doong, S.-H., and Wu, C.-H. Machine Learning. Wiley Encyclopedia of Computer Science and Engineering. John Wiley & Sons Inc. 13 July 2008. Available: http://dx.doi.org/10.1002/9780470050118.ecse228

[5] Lehmusvaara, J. Robust Multi-Class Decision Trees. Master of Science Thesis. Tampere University, Faculty of Information Technology and Communication Sciences, Tampere. 2021. Available: https://urn.fi/URN:NBN:fi:tuni-202112139162

[6] Calzavara, S., Lucchese, C., Tolomei, G., Abebe, S. A. and Orlando, S. Treant: training evasion-aware decision trees. Data Min. Knowl. Discov. 34.5, pp. 1390–1420, 2020. Available: https://doi.org/10.1007/s10618-020-00694-9.

[7] Kingsford, C., Salzberg, Steven, L. What are decision trees? Nature Biotechnology 09.26(9) 1011-3. September 2008. Available: http://dx.doi.org/10.1038/nbt0908-1011

[8] Hastie, T., Tibshirani, R. and Friedman, J. The Elements of Statistical Learning (2nd edition). Springer-Verlag. New York, 2009. Available: https://link-springer-com.libproxy.tuni.fi/book/10.1007/978-0-387-84858-7

[9] Breiman, Leo. Random Forests. Machine Learning; Dordrecht, Vol. 45, Iss. 1. October 2001. Available: http://dx.doi.org/10.1023/A:1010933404324

[10] Zhou, Z-H. Ensemble Methods: Foundations and Algorithms. CRC Press LLC. London, 2012. Available: https://ebookcentral.proquest.com/lib/tampere/detail.action?docID=952003

[11] Freund, Y., Schapire, R. A Short Introduction to Boosting. Journal of Japanese Society for Artificial Intelligence 14(5):771-780. September 1999. Available: http://www.yorku.ca/gisweb/eats4400/boost.pdf

[12] Sharif M, Bauer L, Reiter MK. On the suitability of lp-norms for creating and preventing adversarial examples. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 1605-1613, 2018. Available: https://doi.org/10.48550/arXiv.1802.09653

[13] Kantchelian, A., Tygar, J. D. and Joseph, A. D. Evasion and Hardening of Tree Ensemble Classifiers. Proceedings of the 33rd International Conference on Machine Learning, ICML 2016 New York City, NY, USA. June 19-24, 2016. Available: https://doi.org/10.48550/arXiv.1509.07892

[14] Vos D, Verwer S. Efficient Training of Robust Decision Trees Against Adversarial Examples. International Conference on Machine Learning, pp. 10586-10595, PMLR, July 1, 2021. Available: https://doi.org/10.48550/arXiv.2012.10438

[15] Vos D, Verwer S. GROOT: Growing Robust Trees. 2021. Available: https://github.com/tudelft-cda-lab/GROOT

[16] Ma X, Niu Y, Gu L, Wang Y, Zhao Y, Bailey J, Lu F. Understanding adversarial attacks on deep learning based medical image analysis systems. Pattern Recognition, 110. February 2021. Available: https://doi.org/10.48550/arXiv.1907.10456

[17] Akhtar N, Mian A. Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey. IEEE access. Vol. 6:14410–30. 2018. Available: https://ieeexplore-ieee-org.libproxy.tuni.fi/document/8294186

[18] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E. Scikit-learn: Machine Learning in Python. JMLR 12, pp. 2825-2830. 2011. Available: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[19] Calzavara, S., Lucchese, C., Tolomei, G., Abebe, S. A. and Orlando, S. TREANT: Training Evasion-Aware Decision Trees. 2020. Available: https://github.com/gtolomei/treant

[20] Andriushchenko M, Hein M. Provably robust boosted decision stumps and trees against adversarial attacks. Advances in Neural Information Processing Systems 32. 2019. Available: https://doi.org/10.48550/arXiv.1906.03526

[21] Vanschoren J, Van Rijn JN, Bischl B, Torgo L. OpenML: networked science in machine learning. ACM SIGKDD Explorations Newsletter 15(2), pp. 49-60. June 2014. Available: https://arxiv.org/abs/1407.7722