ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΜΕΛΕΤΗ ΤΕΧΝΙΚΩΝ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ ΓΙΑ ΑΝΙΧΝΕΥΣΗ ΙΟΜΟΡΦΟΥ ΥΛΙΣΜΙΚΟΥ

Διπλωματική Εργασία

του

ΚΩΝΣΤΑΝΤΙΝΟΥ ΚΑΛΑΗ

Επιβλέπων Καθηγητής: Δρ. Πλέσσας Φώτιος

Αναπληρωτής Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

Βόλος, 2020

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΜΕΛΕΤΗ ΤΕΧΝΙΚΩΝ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ ΓΙΑ ΑΝΙΧΝΕΥΣΗ ΙΟΜΟΡΦΟΥ ΥΛΙΣΜΙΚΟΥ

Διπλωματική Εργασία

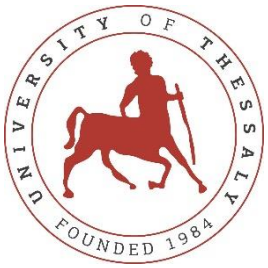
του

ΚΩΝΣΤΑΝΤΙΝΟΥ ΚΑΛΑΗ

Επιβλέπων Καθηγητής: Δρ. Πλέσσας Φώτιος

Αναπληρωτής Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

Βόλος, 2020



UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

MACHINE LEARNING TECHNIQUES FOR HARDWARE TROJAN DETECTION

Diploma Thesis

by

KONSTANTINOS KALAIS

Supervisor: Dr Plessas Fotios

Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly

Volos, 2020

Approved by the Committee on Final Examination:

Advisor	Dr. Plessas Fotios, Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly
Member	Dr. Stamoulis George, Professor, Department of Electrical and Computer Engineering, University of Thessaly
Member	Dr. Sotiriou Christos, Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly

Date Approved: [August 6, 2020]

Ευχαριστήρια

Με την ολοκλήρωση της παρούσας εργασίας, θα ήθελα να ευχαριστήσω τον επιβλέποντα της διπλωματικής μου εργασίας, Αναπληρωτή Καθηγητή κ. Φώτιο Πλέσσα για την πολύτιμη βοήθεια και καθοδήγησή του κατά την διάρκεια της υλοποίησης της εργασίας μου. Θα ήθελα να ευχαριστήσω ολόψυχα τον Κωνσταντίνο Λιάκο, υποψήφιο διδάκτορα του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Θεσσαλίας, ο οποίος μου μεταλαμπάδευσε τις γνώσεις του στο κομμάτι της τεχνητής νοημοσύνης, καθώς και σημαντικές υποδείξεις και επεξηγήσεις που τελειοποίησαν τη παρούσα διατριβή. Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου και τους φίλους μου, που όλα αυτά τα χρόνια δείχνουν την αμέριστη συμπαράστασή τους ως προς το πρόσωπό μου τόσο στο κομμάτι των σπουδών όσο και της ζωής και βρίσκονται πάντα δίπλα μου.

Καλαής Κωνσταντίνος
Βόλος, 2020

ΜΕΛΕΤΗ ΤΕΧΝΙΚΩΝ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ ΓΙΑ ΑΝΙΧΝΕΥΣΗ ΙΟΜΟΡΦΟΥ ΥΛΙΣΜΙΚΟΥ

ΚΩΝΣΤΑΝΤΙΝΟΣ ΚΑΛΑΗΣ

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών,

Πανεπιστήμιο Θεσσαλίας, 2020

Επιβλέπων Καθηγητής: Δρ. Πλέσσας Φώτιος

Αναπληρωτής Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

Περίληψη

Κάθε χρόνο, ο ρυθμός με τον οποίο η τεχνολογία εφαρμόζεται σε τομείς της καθημερινής μας ζωής αυξάνεται με σταθερό ρυθμό. Αυτή η ταχεία ανάπτυξη ωθεί τις εταιρείες τεχνολογίας να σχεδιάσουν και να κατασκευάσουν τα ολοκληρωμένα κυκλώματά τους σε μη αξιόπιστα χυτήρια εξωτερικής ανάθεσης για να μειώσουν το κόστος, δίνοντας έτσι χώρο να αναπτυχθεί μία σύγχρονη μορφή ιού, γνωστή ως ιός δουρείου ίππου συσκευών. Αυτό το είδος ιών αποκτούν πρόσβαση σε κρυπτογραφημένες πληροφορίες, υποβαθμίζουν την απόδοση της συσκευής ή οδηγούν σε πλήρη καταστροφή. Για τη μείωση των κινδύνων που συνδέονται με αυτούς τους ιούς, έχουν αναπτυχθεί διάφορες προσεγγίσεις με στόχο την πρόληψη και την ανίχνευσή τους, με βάση συμβατικές ή Μηχανικής Μάθησης μεθόδους. Ιδανικά, οποιαδήποτε ανεπιθύμητη τροποποίηση σε ένα ολοκληρωμένο κύκλωμα πρέπει να είναι ανιχνεύσιμη με προ-πυριτίου επαλήθευση / προσομοίωση και μετά-πυριτίου δοκιμές. Το μολυσμένο κύκλωμα μπορεί να εισαχθεί σε διαφορετικά στάδια της διαδικασίας κατασκευής, καθιστώντας την ανίχνευση ιών δουρείου ίππου συσκευών μια περίπλοκη διαδικασία. Σε αυτή τη διατριβή, αναπτύξαμε και συγκρίναμε, επτά διαφορετικά μοντέλα Μηχανικής Μάθησης για την ανίχνευση και ταξινόμηση μη-μολυσμένων και μολυσμένων κυκλωμάτων, με βάση τη φάση σχεδίασης και χαρακτηριστικά για κυκλώματα εφαρμογών συγκεκριμένων ολοκληρωμένων κυκλωμάτων. Για την εξαγωγή των χαρακτηριστικών χρησιμοποιήσαμε το επαγγελματικό εργαλείο σχεδίασης, Design Compiler NXT από τη Synopsys. Τα χαρακτηριστικά αποτελούνται από χαρακτηριστικά περιοχής και ισχύος των κυκλωμάτων. Συνολικά χρησιμοποιήθηκαν 50 χαρακτηριστικά περιοχής και ισχύος, που χαρακτηρίζουν τα μολυσμένα κυκλώματα από τα μη-μολυσμένα. Τα πειραματικά αποτελέσματα καταδεικνύουν ότι το μοντέλο ανίχνευσης ιού δουρείου ίππου συσκευών

κάνοντας χρήση του μοντέλου Gradient Boosting (GB) μπορεί να επιτύχει 100% ακρίβεια και F-Measure μετρική. Στο τελευταίο μέρος της διατριβής μας, συγκρίναμε στα ίδια κυκλώματα, το μοντέλο GB με υπάρχοντα μοντέλα της φάσης σχεδίασης και το μοντέλο GB μας επιστρέφει την υψηλότερη ακρίβεια και F-Measure.

MACHINE LEARNING TECHNIQUES FOR HARDWARE TROJAN DETECTION

KONSTANTINOS KALAIS

Department of Electrical and Computer Engineering, University of Thessaly

Supervisor: Dr Plessas Fotios

Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly

Abstract

Every year, the rate at which technology is applied on areas of our everyday life is increasing at a steady pace. This rapid development drives the technology companies to design and fabricate their integrated circuits (ICs) in non-trustworthy outsourcing foundries to reduce the cost, thus, leaving space for a synchronous form of virus, known as Hardware Trojan (HT), to be developed. HTs leak encrypted information, degrade device performance or lead to total destruction. To reduce the risks associated with these viruses, various approaches have been developed aiming to prevent and detect them, based on conventional or machine learning methods. Ideally, any undesired modification made to an IC should be detectable by pre-silicon verification/simulation and post-silicon testing. The infected circuit can be inserted in different stages of the manufacturing process, rendering the detection of HTs a complicated procedure. In this thesis, we developed and compared, seven different Machine Learning models for the detection and classification of Trojan Free and Trojan Infected circuits, based on Gate Level Netlist phase and features for Application Specific Integrated Circuit (ASIC) circuits. For the extraction of the features we used the professional design tool, Design Compiler NXT from Synopsys. The features consist via area and power characteristics of the circuits. In total they were used 50 area and power features, which describe Trojan Infected nets from Trojan Free. The experimental results demonstrate that the HT detection model with Gradient Boosting (GB)-based approach can achieve 100% accuracy and F-measure. In the last part of our thesis, we compared on the same circuits, our GB model with existed GLN models and our GB model returns the highest accuracy and F-measure.

Contents

LIST OF TABLES

LIST OF FIGURES

Chapter 1. INTRODUCTION

1.1 Motivation and preliminaries

Our century has been recognized as the century of technological evolution, and not unjustly. Every year the need for more technological ideas leads, to even more complex and sophisticated software and hardware. Also, it must be mentioned that a significant reason for this technological complexity, is the usage of tiny sensors to even more devices in a combination of Data Analysis (DA) and Artificial Intelligence (AI).

This evolution of technological complexity it was not possible not to affect the design industry. Specifically, design companies are not able to manufacture their own circuits. To overcome this problem, they outsource the design and more frequently the fabrication of their ICs to manufacturing companies, thus risking potential security issues. The newest example of security issues consists of the HTs.

Specifically, design companies are not able to manufacture their own circuits. In order to overcome this obstacle, they outsource the design and more frequently the fabrication of their ICs to manufacturing companies, thus risking potential security issues. Apart from this, Intellectual Property (IP) cores are obtained from various third-party suppliers so as to produce modern and complex ICs with the scope of making security issues even more sensitive. The newest example of security issues consists a new type of virus known as HT.

HTs are viruses which are associated with changes affecting the circuits during the design and/or fabrication phase and are usually introduced by an untrusted design foundry or design software. Due to the complex nature of modern circuits, HTs can be introduced into every IC development phase (Figure 1) and stay dormant until triggered by a wide range of activation mechanisms. HTs are associated with unexpected IC malfunctions, circuit destruction, as well as the leakage of sensitive information regardless of the encryption state [1]. In the recent years in electronics' field, HT infection became a serious issue with chances of being a severe threat from a technological but also social aspect.

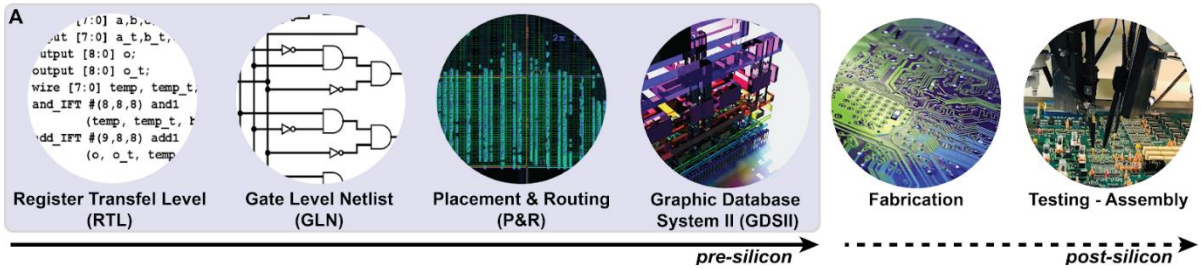


Figure 1: IC development phase

It is naturally created the question about the companies or/and organization that benefit from inserting HT into ICs [2]. For example, the competitor of a company could infect its “inner-buildings” with inserting a virus in the form of circuit into an IC and trying to confine its market share or reduce its profits and consumers’ preference on its products. Another example of affected industries we could mention military issues, HT attacks is a highly concerned threat for many industries. It was first appeared during the Cold War between USA and USSR. At the time, the USSR and USA used the HT to intercept the communication signals of each other. HT has been the subject of academic research a few years ago when the US Department of Defense publicly expressed concerns over the military reliance on ICs manufactured abroad. A recently reported trojan attack involves US Navy, who discovered a hardware ”backdoor” in a microchip used in different industries [1]. As we can understand, such industries with critical information that needs to be under severe security frames, are in a high risk of being ”hurt” so they need to enhance their defense mechanisms. Also, The U.S. Department of Defense has created a “Trusted Foundry Program” to ensure its military equipment remains free of HTs by using only accredited foundries. This means that only American foundries which are located on the American soil and which underwent the strictest vetting process are allowed to work on the chips for the U.S. Department of Defense [3].

The pre and post silicon phases, where IC’s verification-simulation and testing are included respectively, should detect any modification on an IC that is opposite to the desirable conditions. The procedures that pre-silicon phase contains, verification and simulation, needs an integrated model of the entire IC. However, this model could be sometimes unavailable, for example in case where IPs come from third-party suppliers and IC design is based on IP. At the post-silicon phase, we can verify the design of the IC either by comparing the way the circuit functions and its characteristics with a golden model of the circuit [4] [5] [6], or through de-packaging and Reverse Engineering of the creation process of the IC [7]. The process of destructive de-packaging of the IC for its design verification cannot be scalable using the current highest-developed approaches [8]. We cannot detect HTs using Logic Testing through the post-manufacturing phase. The reason can be found at the stealthy nature of HTs and the inordinately vast spectrum of possible trojan instances an adversary can employ. The implementation of the methods for HT detection inherently imposes a trade-off between the desired detection effectiveness and the price designers can afford. These techniques may require extra on-chip circuitry, considerable modifications on post-silicon test setup or both to allow obtaining the internal signals necessary to perform the detection. Therefore, the trojan identification cost is a combination of the required area, test duration and setup charge overhead. The search for alternatives to mitigate this cost is the main challenge in HT detection techniques [9].

The rest of this diploma thesis is divided into five units that take place in Chapters 2-6, respectively. On Chapter 2 we introduce the terminology and definitions of Machine Learning (ML) and the steps need to be followed in order to make predictions from the initial data

collection. On Chapter 3 we analyze the methods we use for the training/validation and the testing of our models. On Chapter 4 we present the final results of our models and in comparison, with the existing work. On Chapter 5 and 6 we referred to the concluded ideas of this thesis and possible future improvements on the current work.

Chapter 2. Machine Learning

2.1 Terminology and definitions of Machine Learning

Humans and computers have few differences, with the most important found on the first ones' ability to learn from their prior experiences, while machinery systems need to be strictly guided to the process of executing a task. Computers are machines absolutely following the logic with lack of common sense. In other words, we must provide them with detailed, step-by-step information on exactly what to do, in order to do something.

Our duty is to write scripts and program computers to follow those instructions. Here is where the area of ML comes in. Its' concept is concentrated on the idea of setting computers to learn from past experiences and data, through the use of computational algorithms and mathematics.

As a subfield of Artificial Intelligence (AI), ML make systems able to learn from experiences-past data without being explicitly programmed. It centers its focus on developing such computer programs which can use data to learn for themselves. We try to collect useful for our scope data and make as good as possible – depending on our model, as we will see later – predictions. The figure below presents a typical form of the ML process.

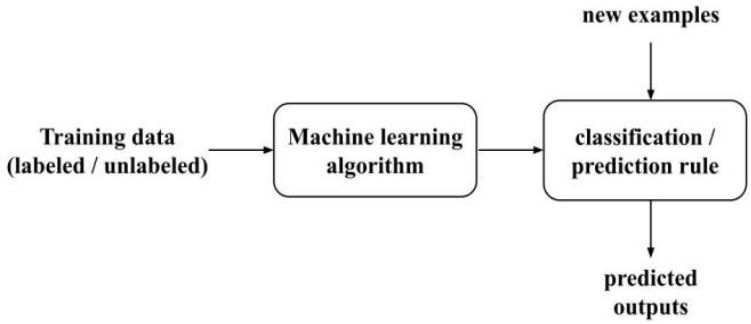


Figure 2: ML training process [10]

2.2 Steps of Machine Learning

The goal of ML is to leak information from data, since data is the link to unlock ML. Below, we will see the steps of ML, from the first stage of data collection until the last step of prediction.

Figure 3: Steps of ML

2.2.1 Data Collection

In the rapid evolution of the information in the century we run, we generate data at an unprecedented rate. That's how the term big data appears in our everyday life. These data can be either numeric (eg height, temperature, population) or categorical (eg color, genres of music). Some of them are either labelled, for supervised learning, or not labelled, for unsupervised learning. The first step of ML consists of collecting data that are required for training the model to provide us with accurate predictions. Data collection could be considered as the process of gathering information from different sources, such as websites, institutions, government's services, etc.

Figure 4: Big Data

2.2.2 Data Preparation

Rough data is not usually useful. The collected data needs to be normalized, prepared, cleaned from the duplicated samples and errors plus bias must be removed. In order to examine if we have collected the right data for our problem or if any data is missing, we can visualize them and check for outliers and patterns.

2.2.3 Choosing a model

The next step of ML includes selecting the best suitable model. Sometimes there are more than one models which can be used for our purpose. The criteria for choosing the best model is to see which one meets our goal. Apart from this, we should know how much time is needed

for preparing the model, its accuracy and scalability. Picking a more complex model does not necessarily means that will achieve the maximum results. On paragraph 1.3 we will examine the most commonly used ML algorithms.

2.2.4 Training

The process of training the model collects the main focus of the ML methodology, since this is the way to alter some parameters and improve the predictions of the model. Each training step consists of updating the weights and the biases. The model is created based on labelled data samples – in supervised ML – and trying to leak inferences from not labelled data – in unsupervised ML.

Figure 5: ML workflow

2.2.5 Evaluation

The next step after the training phase includes testing our model on unseen dataset and see how it performs. The result of an evaluation phase can describe how the model will work on real world data. As the number of real-world variables increases, the training and testing data should be also increased in order to produce better results with the metrics in the evaluation phase.

2.2.6 Parameter Tuning

As the training phase continues, we need to update the weights in order to have better results from cycle to cycle. So, as the number of training steps grows then we can get more accurate results. However, before getting into the training process we should tweak the parameters of the model and experiment with the different results, in order to get the optimal ones.

2.2.7 Prediction

Once you have completed the steps of data collection and preparation, and after we select, train and evaluate the model with the optimum parameters, it is time for using predictions to answer questions regarding our dataset. The word “prediction” sometimes is misleading. It can be used to describe the predictions of a future outcome, but on the other hand it can refer to an action that has already happened and will be examined whether or not it was valid. For example, when transferring money through bank accounts it will guess if the transaction was legitimate.

2.3 Tasks of Learning

ML applications are divided into two main categories: supervised and unsupervised learning. This depends on how the signal of the learning system learns the data patterns and relationship between inputs and outputs. In supervised learning, the data of our problem are created in such way that we can relate the example inputs with corresponding outputs. The goal is to create a general rule that connects inputs with outputs. In other words, the trained model is used to predict the missing labels for the test data. In unsupervised learning, there are unlabeled target values both on training and test sets. The trained model here processed input data so as to find hidden patterns.

2.3.1 Supervised Learning

A supervised ML algorithm [11] takes labeled data for input and creates an output model which can predict future events with provided new data. Such an algorithm could be either used for a classification problem or a regression problem. It starts analyzing a known training dataset and creates a function to produce predictions about the output values. We divide the input dataset into training and test set. The training set has output values that we will classify or predict. The algorithms learn recognition patters from the train set and use them for classifications or predictions in the test set [6].

2.3.2 Unsupervised Learning

This type of learning deals with non-labeled data, or data that had not been divided into categories. It derives information about the features from the data and when it meets new data, it uses the already learned features in order to classify the new samples into a group of values, known as class. The main areas that unsupervised learning is used are clustering and dimensionality reduction. As for the first one, a sample of one group will have same properties to the other data in its group, so as to when it is compared with a sample of another group, they should have much different properties. Feature reduction compresses the data by simply reducing the dimension of the feature set. Specifically, random variables are removed until

the dataset did not lose its structure and important information. The feature set could contain hundred columns – features, let's say for example the data points make up a sphere of the three dimensional space, and we want to reduce the dimension to two, in our example to a two dimensional circle [12]. In that way, we can improve the general model performance since the data are now easier to be stored, run computations on them and visualize them.

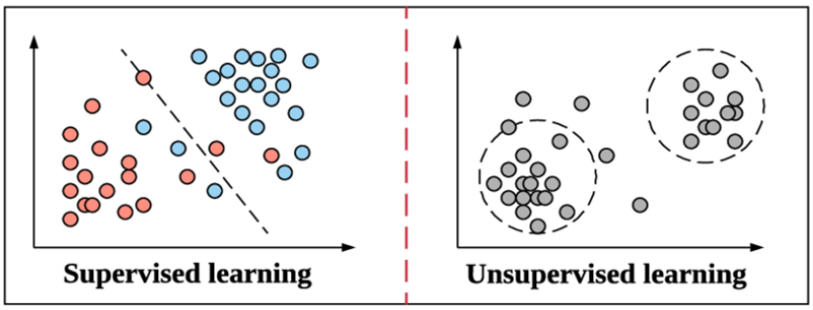


Figure 6: Supervised vs Unsupervised Learning

2.4 Analysis of Learning

Dimensionality reduction (DR) is a technique that appears in both applications of supervised and unsupervised learning. Its usage appears on reducing the number of random variables under consideration, by obtaining a set of principal variables. It is usually performed prior to applying a classification or regression model in order to avoid the effects of dimensionality. Some of the most common DR algorithms are the following: (i) principal component analysis [13], (ii) partial least squares regression [14], and (iii) linear discriminant analysis [15].

2.5 Learning Models

Below, we analyze the most common learning models used on the works that are presented in this thesis and focus mainly on the ones that we used for our problem.

2.5.1 Regression

Regression belongs to the supervised learning models and tries to predict an output variable according to the known input variables. Some of the most known algorithms are: linear regression [16], logistic regression (LR) [17] and stepwise regression [18]. Also, there are available more complex algorithms, such as ordinary least squares regression [19], multivariate adaptive regression splines [20], multiple linear regression, cubist [21], and locally estimated scatterplot smoothing [22].

2.5.1.1 Logistic Regression

Logistic Regression (LR) is a supervised ML algorithm used for classification problems, and specifically for categorizing observations into a group of discrete classes. Although linear regression assigns observations to a continuous number of values, LR applies on its output a transformation - activation – function, called the logistic sigmoid function. In that wat it returns a probability value which can then be matched with two or more classes. LR is used when the target – dependent - variable is categorical. For example, to predict whether an email is spam (1) or not (0) (binary LR) or to predict whether a car with specific characteristics belongs to a model like BMW, Mercedes, Ford, etc (multiclass LR).

Binary LR: Suppose we are given a dataset that includes student exam results and we want to predict if a student will pass or fail the exam according to the combination of number of hours he has slept and hours spent for studying. In our data, we have to deal with the features of slept hours and studied hours and only two classes: passed (1) and failed (0), as a result of the pass or fail of a student's result. Graphically we could represent these data with a scatter plot (Figure 8).

Studied	Slept	Passed
4.85	9.63	1
8.62	3.23	0
5.43	8.23	1
9.21	6.34	0

Figure 7: The first four samples of these data

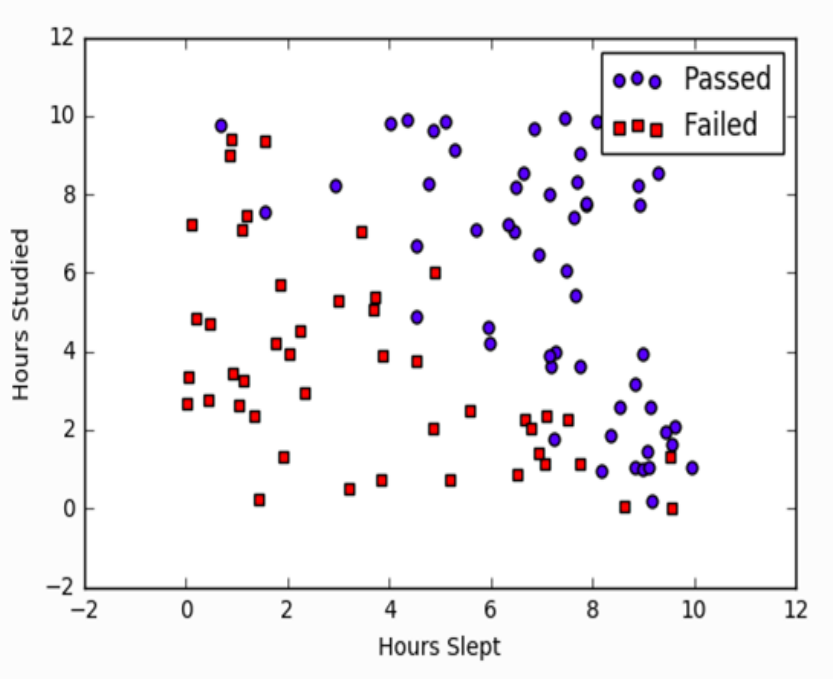


Figure 8: Scatter plot of the toy-example data

Sigmoid activation: In ML, we make use of the sigmoid function so as to match predicted values with the probabilities. The function converts any real value into a probability, which is in fact another value between 0 and 1.

- s(z) = output between 0 and 1 (probability estimate)
- z = input to the function (your algorithms prediction e.g. ax + b)
- e = base of natural log

$$S(z) = \frac{1}{1 + e^{-z}}$$

Decision Boundary: The sigmoid function returns the predicted value of the probability in the range between 0 and 1. In order to map this to a value that belongs to a discrete class (true/false,

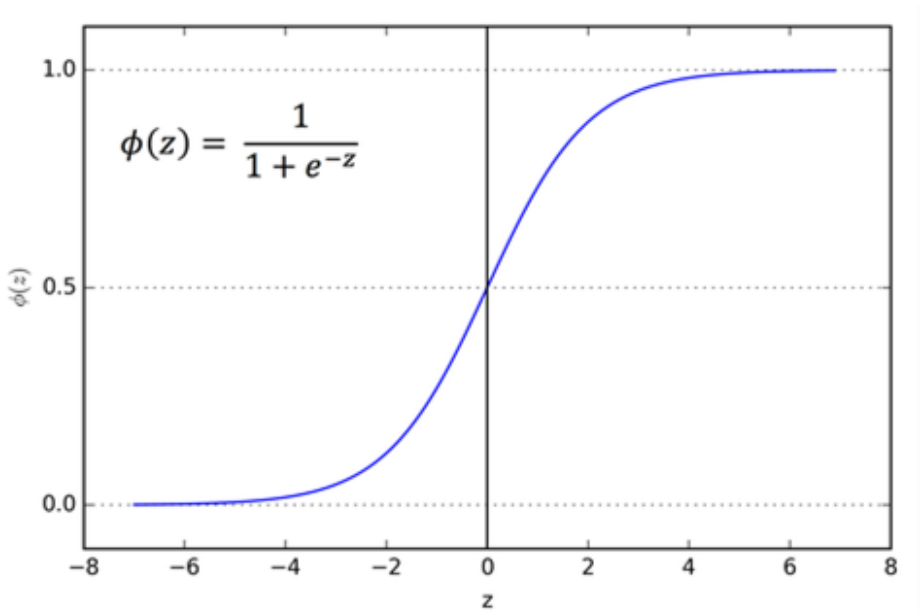


Figure 9: Sigmoid function

BMW/Ford/etc), we select a threshold value below which we will classify values into the second class (in the case of binary LR) and above which we will classify values into the first class. The same concept is followed when we have more than two discrete classes (multiclass LR).

$$\begin{cases} p \geq 0.5, for\ class = 1 \\ p < 0.5, for\ class = 0 \end{cases}$$

For example, if our tipping point (threshold value) was 0.5 and our prediction function returned 0.8, we would classify this observation to the class 1 (positive value). If our prediction was 0.3, we would classify the observation to the class 2 (negative value). In the case of multiclass LR where we have to decide between multiple classes to map the observation, we could map the observation to the class with the highest predicted probability.

We can create now a prediction function by taking into consideration our knowledge of decision boundaries and sigmoid functions. Such a function could compute the probability of our observation for being positive. In other words, the probability of our observation belongs to the class 1 and its notation is P(class=1). As the probability increases and reaches 1, our model becomes more stable and we could be more confident that the observation is in class 1. On the other side, as the probability decreases and gets far away from 1, our model gives a higher probability that the observation is in class 2.

$$P(class = 1) = \frac{1}{1 + e^{-z}}$$

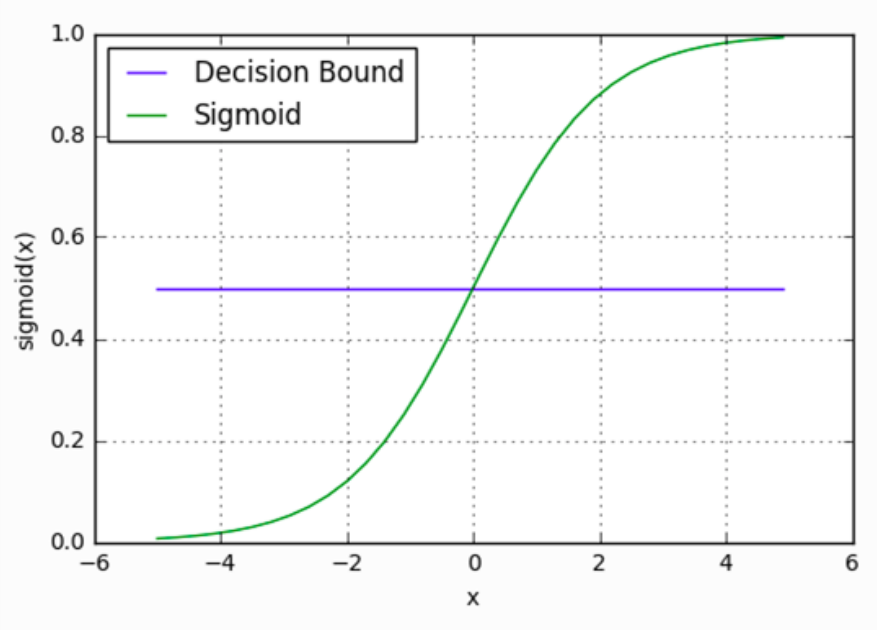


Figure 10: Decision boundary

So, now using the sigmoid function we will apply the transformation to our output and it will return a probability value between 0 and 1: z = w_0 + w_1 Studied + w_2 Slept.

The final step is to map the predicted probabilities to the discrete classes with labels 0 or 1. So, we classify our sample at class with label = 1 if the probability ≥ 0.5, else at class with label = 0.

2.5.2 Clustering

Clustering [23] constitutes an unsupervised learning model and is used to find groupings of data points (clusters). Data points of the same group have same characteristics and are dissimilar to the data points in other points. Most known clustering techniques are developed, such as the k-means clustering [24], the hierarchical technique [25], and the expectation maximization technique [26].

2.5.3 Bayesian Models

Bayesian models (BM) are statistical models where you use probability to represent all uncertainty within the model, both the uncertainty regarding the output but also the uncertainty regarding the input (parameters) to the model. This type of model belongs to the supervised learning category and is mainly used for classification or regression problems. Well

established Bayesian algorithms are Naive bayes [27], gaussian naive bayes, bayesian network [28], mixture of gaussians [29], and bayesian belief network [30].

2.5.4 Instance Based Models

Instance based models (IBM) belong to a family of techniques for classification and regression, which produce a class label based on the similarity of the query to its nearest neighbors in the training set. They do not create an abstraction from specific instances. The disadvantage of these models is that their complexity grows with data. The most common learning algorithms in this category are the k-nearest neighbors (KNN) [31], locally weighted learning [32], and learning vector quantization [33].

2.5.4.1 KNN

The KNN is a supervised ML algorithm useful for solving regression and classification problems in a simple and easy way. The KNN algorithm is based on the assumption that same things exist in a close area. In other words, similar things are close to one another.

KNN is based on the idea of similarity (also known as distance, proximity, or closeness) figuring the space between points on a graph. There are various methods of calculating distance, and one way might be preferable depending on the problem we are solving. For our problem, we use the most common distance metric, which is the Euclidean distance between two points x_1 and x_2, and is given by the formula: $\mathrm{d}(\mathrm{x}_1 - \mathrm{x}_2) = ||\mathrm{x}_1 - \mathrm{x}_2||_2$.

We will see now who does the KNN algorithm work. First of all, we have to load the data - usually called X - and their target values – usually called y - we want to classify. Then we initialize K to a preferable number of neighbors and for each data sample we compute the distance between the sample whose target value we want to classify and the current sample from the data. We add both the index and the distance of the query example to an ordered list of indices and distances and sort this list in an ascending order (from smaller to bigger), with the distance as order criteria. Finally, we pick the first K entries from the sorted list and get the labels of the selected K entries, so we can return the form of the K labels.

For example, as we can see in the figure below for K=3 the tested sample would have been classified to class B, and for K=7 would have been classified to class A.

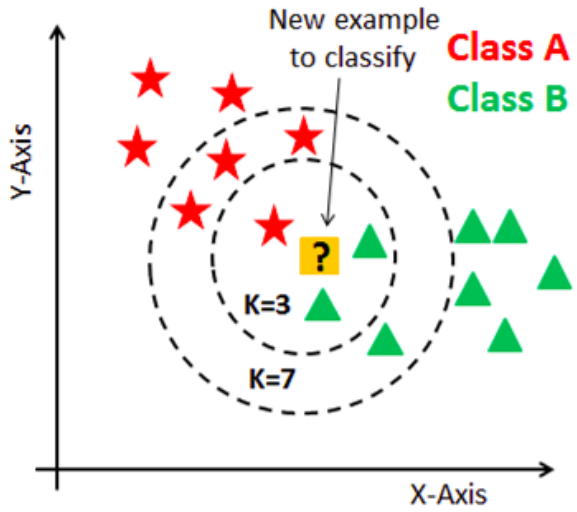


Figure 11: KNN example

In order to choose the best number for the variable K to suit with our problem, we have to execute the KNN algorithm sometimes with different values of K and choose the K that decreases the number of errors we meet while keeping up the ability of the KNN algorithm to make accurate predictions when it is provided with data that has never seen before. As the value of K is reduced to one, our predictions' stability is also decreased. On the other, as we raise K's value, our predictions' stability is increased to because of the majority voting in the compared samples. As a result, it is more possible to increase - until a specific level- the percentage of accuracy for our predictions.

2.5.5 Decision Trees

Decision trees (DT) are models with tree-like architecture and are typically used for classification or regression problems. The tree is organized by splitting the input dataset into smaller subsets based on an attribute value test. We have to start from the root of the tree in order to make a prediction. We compare the values of the root attribute with the target value's attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node until we reach leaf nodes that represent the final decision. Some of the basic ML algorithms that depend their operation on DT are classification and regression trees [34], the iterative dichotomiser [35] and the chi-square automatic interaction detector [36].

2.5.6 Artificial Neural Networks

Artificial neural networks (ANNs) are classified into two main categories: "Traditional ANNs" and "Deep ANNs".

ANNs take inspiration from the functionality of the human brain, imitating more complex functions like pattern recognition and decision making [37]. Billions of neurons create the structure of human brain, by communicating and processing any information provided. In a similar way, an ANN as it constitutes a simplified version of the biological neural network's structure, is consisted of interconnected units that processes information and are organized in a specific topology. An ANN contains numerous nodes that are arranged in multiple layers, including the following ones:

1. An initial layer that accepts the input data and fed the system with information,
2. Various hidden layers (one or more) for the learning process, and
3. An output layer for the final prediction's value.

ANNs belong to the category of supervised learning algorithms. Some of the common learning algorithms used in ANNs are perceptron algorithms [38], the radial basis function networks [39], back-propagation [40], and resilient back-propagation [41]. Apart from those, a large number of ANN-based learning algorithms have been used on previous researches, such as multilayer perceptron (MLP) [42], counter propagation algorithms [43], adaptive-neuro fuzzy inference systems [44], autoencoder, XY-Fusion, supervised Kohonen networks [45], as well as Hopfield networks [46], extreme learning machines [47], self-organising maps [48], generalized regression neural network [49], ensemble neural networks or ensemble averaging, and self-adaptive evolutionary extreme learning machines [50].

Deep ANNs are usually referred to the field of deep learning (DL) – a relative new area of ML research - and mainly called as deep neural networks (DNNs) [51]. They deal with computational models that consist of multiple processing layers and help them find data patterns and representations using multiple abstractive levels. DL has the advantage that sometimes the model itself perform the process of feature extraction. DL have lately huge impact on many different areas and industries, such as the Hardware Trojan detection field. DNN's could be considered as a simple ANN with additional hidden layers between input and output layers, and could have use on both supervised and unsupervised problems. A very common type of DL model is the convolutional neural network (CNN), where convolutions are performed in the image domain and produce the extraction of feature maps. A brief introduction on CNNs is given in the literature [52]. Other typical DL architectures include deep Boltzmann machine, auto-encoders [53] and deep belief network [54].

2.5.6.1 MLP

As we mentioned before, Neural networks are built as the model of neurons present in the human brain. Neural networks are typically stacked layers of interconnected neurons or nodes each having an activation function. The idea is to give our network several inputs (each input is a single feature). Then, we navigate through the network by applying weights to our inputs,

summing them, and finally using an activation function to obtain the subsequent nodes. This process is repeated multiple times through the hidden layers, before applying a final activation function to obtain the output layer, our prediction. An example with n input nodes and 1 output node is shown below, with the step activation function:

$$f(x) = \begin{cases} 0, if\ x < 0 \\ 1, if\ x \geq 0 \end{cases}$$

The training takes place on the weights, which are initialized randomly. They are updated by trying to minimize training error using gradient descent, and are used after the training phase to make predictions.

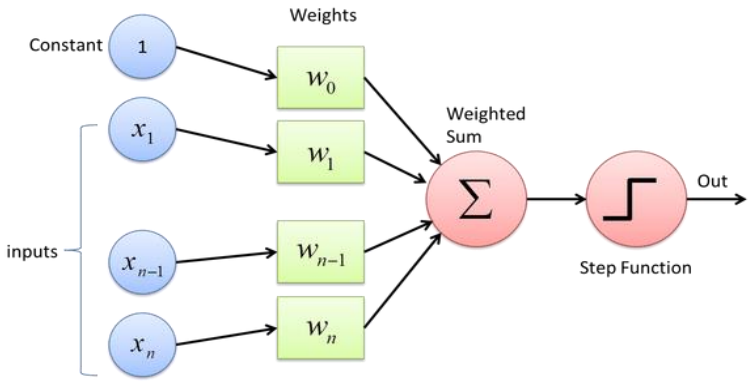


Figure 12: MLP's structure

Gradient Descent: Let's say we have an example of a regression problem, so for given input x we want to predict the output. We present the hypothesis as:

$$h_\theta(x) = \theta_0 + \theta_1$$

, where ϑ_0 and ϑ_1 are the parameters.

Then we need a function for minimization these parameters over our dataset. One common function that is often used is mean squared error, which measure the difference between the estimator (the dataset) and the estimated value (the prediction). It looks like this:

$$MSE = \frac{1}{n} * \sum_{i=1}^{n} (Y_i - Ypred_i)^2$$

It turns out we can adjust the equation a little to make the calculation down the track a little simpler. We end up with:

$$\frac{1}{2m} * \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

I will refer to the cost function as **J**(ϑ). In our problem we will need to minimize a cost function, called Mean Squared Error (MSE). The use of Gradient Descent function is valuable for minimizing it, by changing the theta values, or parameters, step per step, until we hopefully achieved a minimum.

We start by initializing ϑ_0 and ϑ_1 to any two values, say 0 for both, and go from there. Formally, the algorithm is as follows:

$$\theta_j = \theta_j - \ a * \frac{d}{d\theta_i} J(\theta_0, \theta_1) \quad (for\ j = 0\ and\ j = 1)$$

and repeat until convergence:

$$\theta_0 = \theta_0 - \ a * \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

$$\theta_1 = \theta_1 - \ a * \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) * \ x^{(i)}$$

, where α is the learning rate, or how quickly we want to move towards the minimum.

2.5.7 Support Vector Machines

Support vector machines (SVMs) were first introduced in the work of [55] on the foundation of statistical learning theory. SVMs are supervised learning models used for data analysis and pattern recognition in classification and regression problems. SVMs transforms the input feature space into higher-dimensional feature space using the kernel trick dot product, where (x, y) is replaced with k (x, y) = <f(x), f(y)> (f is called kernel function). We can find each dataset's sample distance to a given dividing hyperplane. We call margin the minimum distance from the samples to the hyperplane. The transformed data can be separated using a hyperplane, the dividing curve between distinct classes. The optimal hyperplane maximizes the margin.

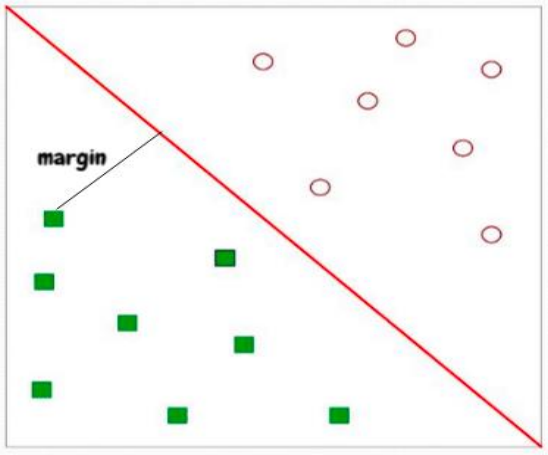


Figure 13: Hyperplane in 2d feature space

Based on global optimization, SVMs deal with overfitting problems, which appear in high-dimensional spaces, making them appealing in various applications [56] [57]. Most used SVM algorithms include the support vector regression [58], least squares SVM [59] and successive projection algorithm-SVM [60].

In other words, an SVM is a linear separator that focuses on creating a hyperplane with the largest possible margin. Its goal is to classify a new sample by simply computing the distance from the hyperplane. On a two-dimensional feature space, the hyperplane is a single line dividing the two classes (as we can see on Figure 13). On a multi-dimensional feature space, where the data are non-linearly separable an SVM cannot linearly classify the data. In this case it uses the kernel trick. The main concept has to do we the fact that the new multidimensional feature space could have a linear decision boundary which might not be linear in the original feature space. This can be done using various kernel functions such as:

- Polynomial kernel, popular in image processing. Equation is: $k(x, y) = (x * y + 1)^d$, where d is the degree of the polynomial
- Gaussian kernel, used when there is no prior knowledge about the data. Equation is:

$$k(x, y) = e^{(-\frac{||x-y||^2}{2*\sigma^2})}$$

  , where σ is standard deviation.
- Gaussian radial basis function (RBF), also used when there is no prior knowledge about the data. Equation is: $k(x, y) = e^{(-\gamma||x-y||^2)}$, where $\gamma = \frac{1}{2*\sigma^2}$
- Hyperbolic tangent kernel, which can be used in neural networks. Equation is: *k(x, y) = tanh(kx · y + c), for some k, c > 0*

2.5.8 Ensemble Learning

For some purposes it is more effective to combine different individual learners – like classifiers - into one learner in order to produce better prediction results for classification, regression or other tasks (Figure 14). It is observed that by collecting various learners is better at executing a task than individual learners. Ensemble Learning (EL) methods – as a divide and conquer approach - help to minimize the causes of error in the machine learning models, such as noise and bias. Decision trees have been typically used as the base learner in EL models, for example, random forest (RF) [61], whereas a large number of boosting and bagging implementations have been also proposed, for example, boosting technique [62], adaboost [63] and bootstrap aggregating or bagging algorithm [64].

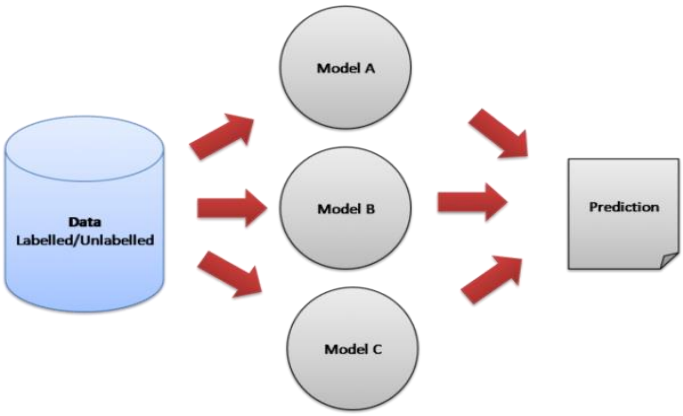


Figure 14: Ensemble Learning

2.5.8.1 RF

A decision tree is a graphical representation of data which applies a branching method to describe with illustrations any possible result of a decision. Each tree branch stands for a possible option that is available for taking a decision. The depth of the tree is extended as new decisions need to be made. So, the decision tree as a tree structure, has few components; internal nodes for a test on an attribute, the branches that represents a test result and leaf nodes (terminal nodes) that contains information for the class labels. The outcome of a test - based on the node's attributes creates a new branch, and as we go from root to the next node and so on, we reach a leaf node, that belongs to a unique label (class).

As we can see in the figure below, we have to decide for example if a person is fit. Following a series of Boolean tests, we will end up on a" fit" or" Unfit!" leaf node, concluded for the fitness's level of a person.

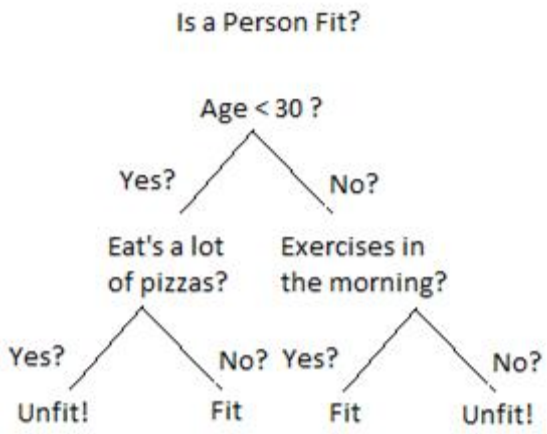


Figure 15: Decision Tree toy example

RF is a summation of Decision Trees. The general idea of this technique is that a mixture of learning models raises the general result. RF builds multiple decision trees and merges them together to achieve the preciseness and stability of the prediction. In that way, it prevents overfitting by creating random subsets of the features, building smaller trees using these subsets and combines them so as to increase the overall performance. With overfitting we mean the result of an overly complex model with too many parameters. If a model is over fitted, we can result that it is inaccurate because its behavior does not represent real data information.

As we can see in the next figure, a simplified RF could categorize a sample to the class with the maximum" votes" among each subtree. RF makes the model more random, while developing the trees. Rather than looking for the most significant feature while splitting a node, it scans for the best element among a random subset of features. This outcomes in a wide variety that by and large results in a greater model.

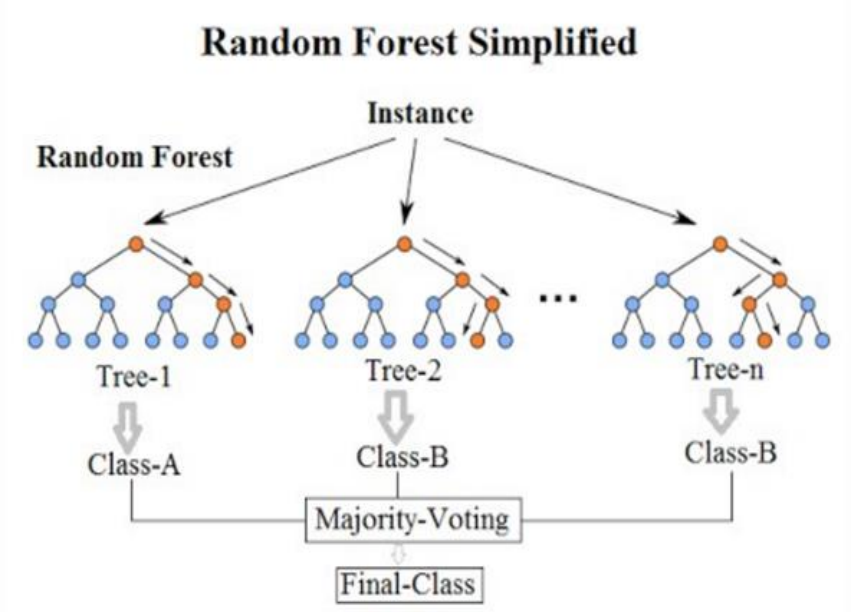


Figure 16: RF simplified

2.5.8.2 Gradient Boosting

Gradient boosting (GB) as a ML method for classification and regression problems is trying to create a model for predictions in the form of a mixture of inefficient prediction models that are called decision trees. In the boosting phase, every new tree is a fit on an altered version of the original data set. Firstly, the GB trains a decision tree and assigns each observation an equal weight. After the first tree assessment, we lower the weights for the observations that are easy to classify and increase the weights of those that are hard to classify. Then, we grow the next tree on this weighted data where we try to improve the predictions of the first tree. Our new model is tree 1 together with tree 2. Then we compute the classification error from this combination of two-tree model and create a third one to predict the revised residuals. We repeat this process for a specified number of iterations. Subsequent trees help us to classify the non-well categorized observations by the former trees. The predictions of the final ensemble model is the weighted sum of the predictions made by the previous three models.

2.5.8.3 XGBoost

XGBoost (XGB) belongs to the family of the ensemble learning methods. Sometimes, it could be insufficient to depend on the results of only one ML method applied to our data. Ensemble learning techniques use a systematic method to combine the predictive power of various learning methods. The output of this combination is a model that provides the totaled result from smaller-weaker- models. Most of the times, we use XGB algorithm with decision trees.

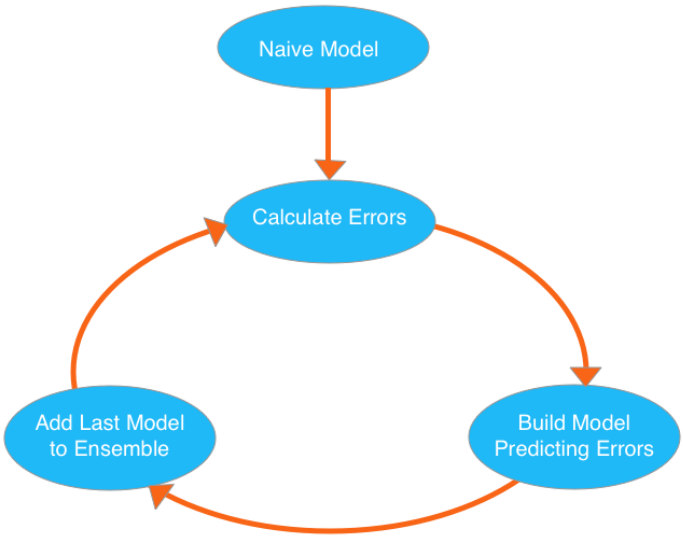


Figure 17: Gradient Boosted Trees diagram

XGB is the main model for dealing with the type of data you store in Pandas DataFrames, known as tabular data, and no other types of data like images and videos. XGB belongs to the Gradient Boosted Decision Trees algorithms, which we will examine later on.

The basic concept follows the circular rule where the method repeatedly creates new models and blends them into an ensemble model. At the beginning of this procedure, we calculate the errors for each observation in the data. Then we create a new model to predict those errors. We add the error-predicting results of this model to the total of the model. Then we add the predictions from all previous models in order to make a new prediction. We can use the added predictions to calculate new errors, create the next model, and add it to the ensemble.

In the boosting phase, each tree is built in a subsequent way, trying to reduce the residual errors of the previous trees. The learning procedure follows the previous tree and updates the remaining errors. The next growing tree will learn from the last version of the residual errors.

The boosting method in its basis has learners with high bias and low - bit higher than random predictions - predictive strength. Each of these base learners produce crucial information that contributes to the ensemble model for the predictive power. As a result, by combining these base-weak learners the overall boosting method develops strong predictive power and becomes a strong learner (with lower bias and variance).

In the boosting technique, trees with fewer splits are usually used, and as a result such small trees can be easily managed and interpreted. On the other side, the bagging techniques such as RF examine trees that are extended to the maximum of their depth. So, it is preferable to go with a boosting instead of a bagging algorithm, since it is an updated version of the last

one. Also, we have to select optimally the stopping criteria since a huge number of trees can cause overfitting. Applying validation techniques like k-fold cross validation we can carefully choose parameters such as the depth of the trees, the number of splits per level, etc.

The boosting phase consists of three steps:

- An initial model F_0 is defined to predict the target variable y. This model will be associated with a residual $(\mathrm{y} - F_0)$
- A new model h_1 is fit to the residuals from the previous step
- Now, F_0 and h_1 are combined to give F_1, the boosted version of F_0. The mean squared error from F_1 will be lower than that from F_0: $F_1(x) < -F_0(x) + h_1(x)$

To improve the performance of F_1, we could model after the residuals of F_1 and create a new model F_2:

$$F_2(x) < -F_1(x) + h_2(x)$$

This can be done for m iterations, until residuals have been minimized as much as possible:

$$F_m(x) < -F_{m-1}(x) + h_m(x)$$

Here, the additive learners do not disturb the functions created in the previous steps. Instead, they impart information of their own to bring down the errors.

As the first step, the model should be initialized with a function F_0(x). F_0(x) should be a function which minimizes the loss function or MSE (mean squared error), in this case:

$$F_0(x) = argmin_\gamma \sum_{i=1}^{n} L(y_i, \gamma) = argmin_\gamma \sum_{i=1}^{n} (y_i - \gamma)^2$$

Taking the first differential of the above equation with respect to y_i, it is seen that the function minimizes at the mean. So, the boosting model could be initiated with:

$$F_o(x) = \frac{\sum_{i=1}^{n} y_i}{n}$$

Chapter 3. Methodology

3.1 Tools used

The code was arranged around Spyder. Spyder is a powerful scientific environment written in Python, for Python, and designed by and for scientists, engineers and data analysts. It features a unique combination of the advanced editing, analysis, debugging, and profiling functionality of a comprehensive development tool with the data exploration, interactive execution, deep inspection, and beautiful visualization capabilities of a scientific package. Furthermore, Spyder offers built-in integration with many popular scientific packages, including Pandas, Ipython, QtConsole, Matplotlib, SymPy, and more. In our work, we mostly make use of Scikit-learn which is a free ML library for Python. It features various algorithms SVM, RF, and KNN, and it also supports Python numerical and scientific libraries like NumPy and SciPy. We can use separate working areas for the Variables values, the command line and code editor.

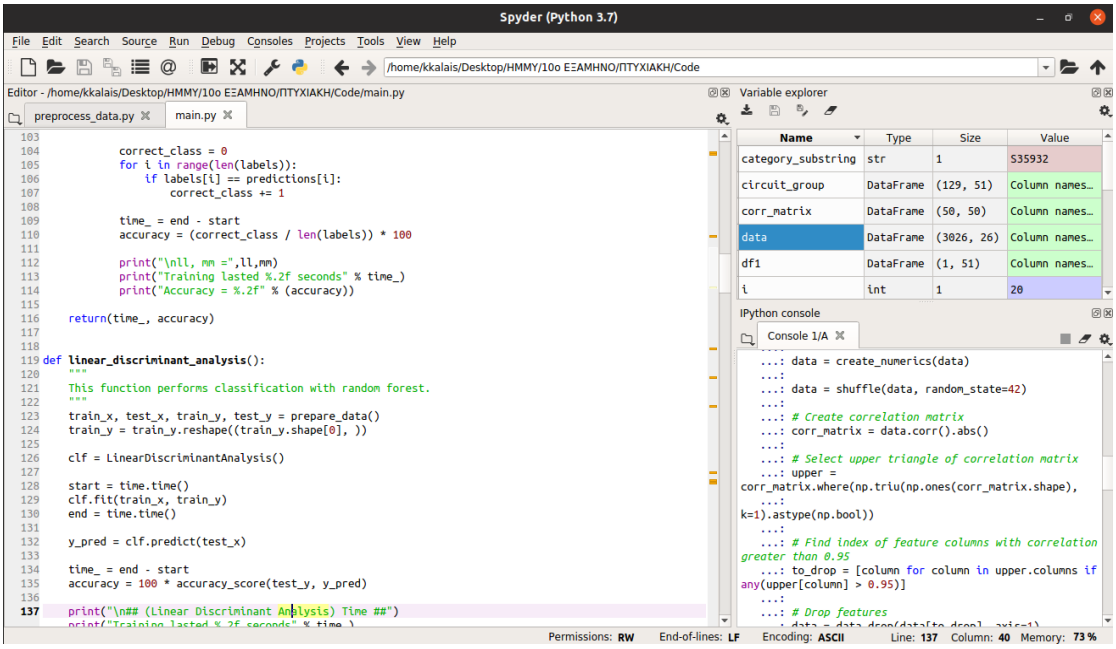


Figure 18: Spyder workspace

3.2 Dataset Construction

Experiments were carried out on a dataset that was created by research benchmarks from Trust-Hub, a certified public library with Trojan free (TF) and Trojan Infected (TI) circuits. For the creation of our dataset we used all the circuits from Trust-Hub, approximately 1.000 circuits. Twenty-one (21) of them refers to TF circuits and all the other refers to TI circuits. Then, using industrial circuit design tools (DC compiler Synopsys) and our own scripts, we

created the dataset for the design phase known as Gate Level Netlist (GLN). We consider dealing with characteristics about the area, power and time of each circuit. The feature extraction process created the final dataset, consisted of 50 characteristics.

3.3 Dataset Preprocessing and Feature Extraction

The dataset had some minor inconsistencies such as missing features that we removed during the dataset cleaning phase. It includes 50 features and after dropping the highly correlated ones - with at least 95% correlation, in order to make our algorithm more cost and time efficient - it consists of the following 24 features, plus the feature with the target values:

Area	Power (1)	Power (2)	Power (3)
Number of ports	Net Switching Power	Cell Internal Power	Black_Box Total Power
Number of sequential cells	Cell Leakage Power	Memory Switching Power	Clock_Network Internal Power
Number of macros/black boxes	IO_Pad Internal Power	Memory Leakage Power	Clock_Network Switching Power
Number of references	IO_Pad Switching Power	Memory Total Power	Clock_Network Leakage Power
Macro/Black Box area	IO_Pad Leakage Power	Black_Box Internal Power	Sequential Internal Power
	IO_Pad Total Power	Black_Box Switching Power	Sequential Leakage Power
	Memory Internal Power	Black_Box Leakage Power	

Table 1: Features set

As we see above, there were only 21 out of 1.000 TF labeled instances, with a particular type of circuit. In order to balance the ratio between TF and infected, we used a reproduced technique in order to reproduce each TF multiple times, to match the total number of TI of the same circuit category. This type of technique is not the optional, but recommended under unbalanced datasets.

As for the features dropping step, when we examine the features of a dataset, some of them might not be useful to make the necessary prediction. Correlation refers to how close two variables (features for our problem) are to having a linear relationship with each other. Features with high correlation are more linearly dependent (e.g. $y = 3 \cdot x$) and hence have almost the same effect on the dependent variable, in contrast with two features that are non-linearly dependent (e.g. $y = x^2$). So, when two features have high correlation, we can drop one of the two features.

We use the LabelEncoder class provided by Scikit Learn library, in order to transform the nominal values (numeric codes used for labelling or identification, such as strings, datetimes, etc) to numeric and enable the algorithm to perform arithmetic operations on these values. Next, we create the correlation matrix, which is computed as follows:

Correlation Matrix = $[Cor(x_j, x_k)]_{ik}$, where x_j and x_k are the vectors of two features' values and

$$\mathrm{Cor}(x_j, x_k) = \sum_{i=1}^{n} \left(x_{ij} - mean(x_j)\right) * \left(x_{ik} - mean(x_{ik})\right) * \left(\sum_{i=1}^{n} \left(x_{ij} - mean(x_j)\right)^2\right)^{-\frac{1}{2}} * \left(\sum_{i=1}^{n} \left(x_{ik} - mean(x_k)\right)^2\right)^{-\frac{1}{2}}$$

and visualize it through the correlation heat-map, as we can see on the figure below.

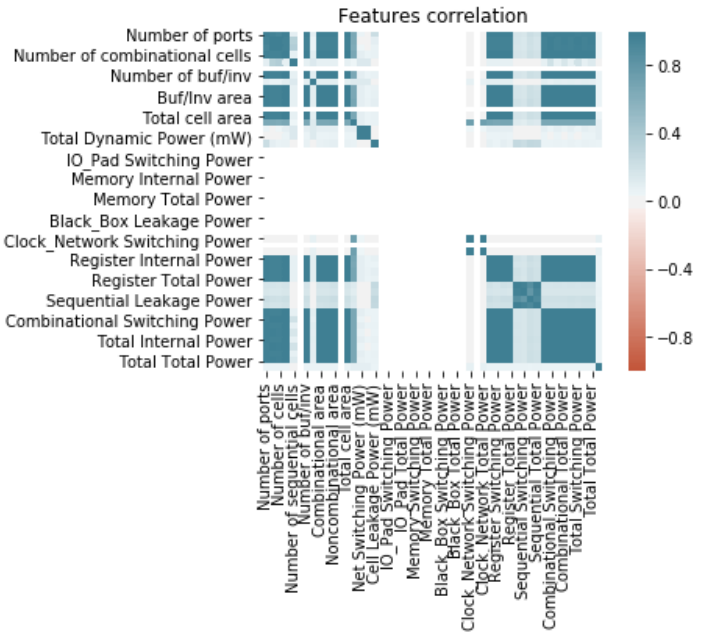


Figure 19: Plot of features correlation

Next, we need to shuffle the data before each iteration. During the training, it is important to shuffle the data so that the model does not learn specific pattern of the dataset. If the model learns the details of the underlying pattern of the data, it will have difficulties to generalize the prediction for unseen data. This is called overfitting. The model performs well on the training data but cannot predict correctly for unseen data.

Lastly, we scaled our data using the MinMaxScaler class of Scikit Learn in the range of (0,1) for easier convergence. MinMaxScaler is a good choice if we want our data to have a normal distribution or want outliers to have reduced influence. Specifically, if max(X) and min(X) are the maximum and minimum values that appear in all dataset for a given feature X, a value X is replaced by newX using the equation:

$$newX = \frac{X - min\ (X)}{max(X) - minx(X)}$$

This procedure was not implemented for the SVM methodology, as SVMs are scale invariant.

3.4 Algorithms used

Based on the previous research papers, we decided on the use of eight algorithms to focus on the issue and we will analyze below the work implemented in each one:

3.4.1 MLP
3.4.2 RF
3.4.3 SVM
3.4.4 GB
3.4.5 KNN
3.4.6 LR
3.4.7 XGB

Table 2: ML algorithms used

3.4.1 MLP

In our case, we construct a probability distribution of the 2 possible values (0 for Trojan-free, 1 for Trojan-infected) by running the outputs through a softmax activation function.

Simply speaking, the sigmoid function only handles two classes, which is not what we expect. The softmax function squashes the outputs of each unit to be between 0 and 1, just like a

sigmoid function. But it also divides each output such that the total sum of the outputs is equal to 1.

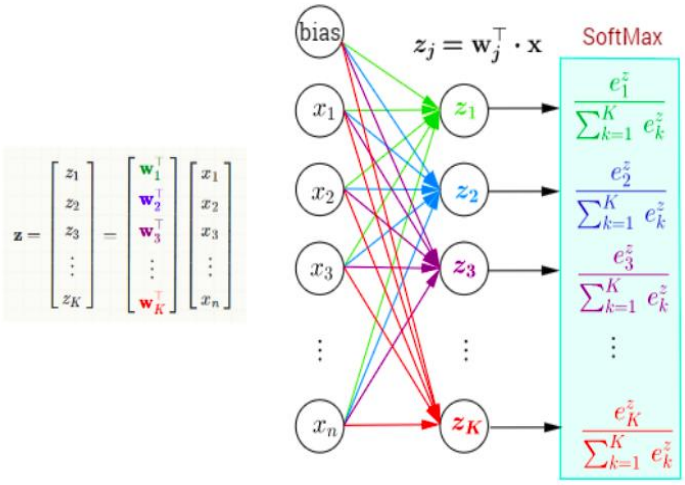


Figure 20: Softmax activation function

The output of the softmax function is equivalent to a categorical probability distribution, it tells you the probability that any of the classes are true. Mathematically the softmax function is shown below, where z is a vector of the inputs to the output layer (if you have 10 output units, then there are 10 elements in z). And again, j indexes the output units, so j = 1, 2, ..., K

$$\sigma(j) = \frac{e^{w_j^T * x}}{\sum_{k=1}^{K} e^{w_j^T * x}} = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

The model was compiled with Adam optimizer, since it is appropriate for problems with very noisy/or sparse gradients, and a categorical cross-entropy loss function, since we have multiple classes, and trained in batches of 150 instances for 50 iterations. For the fully connected neural networks architecture we used:

- A Dense input layer of 15 neurons and rectifier linear unit activation function

$$ReLU(x) = \begin{cases} 0, if\ x < 0 \\ x, if\ x \geq 0 \end{cases}$$

- A Dense hidden layer of 75 neurons and rectifier linear unit activation function
- A Dense output layer of 2 output classes and softmax activation function

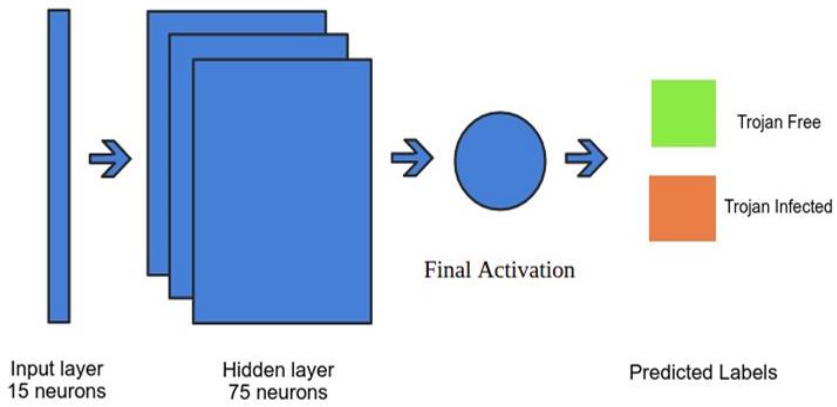


Figure 21: Neural Network

To create our fully-connected neural network, we begin with creating a Sequential model, a linear stack of 3 layers (input, hidden and output as mentioned above). Before training the model, we need to configure the learning process, which is done via the compile method. And then we train our model based on our training data, so as to learn our weights.

After trying various numbers of hidden units in the hidden layer, we conclude that the best accuracy is achieved with 100 neurons. However, we choose a hidden layer of 75 neurons since in that situation we have a good percentage and lower execution time.

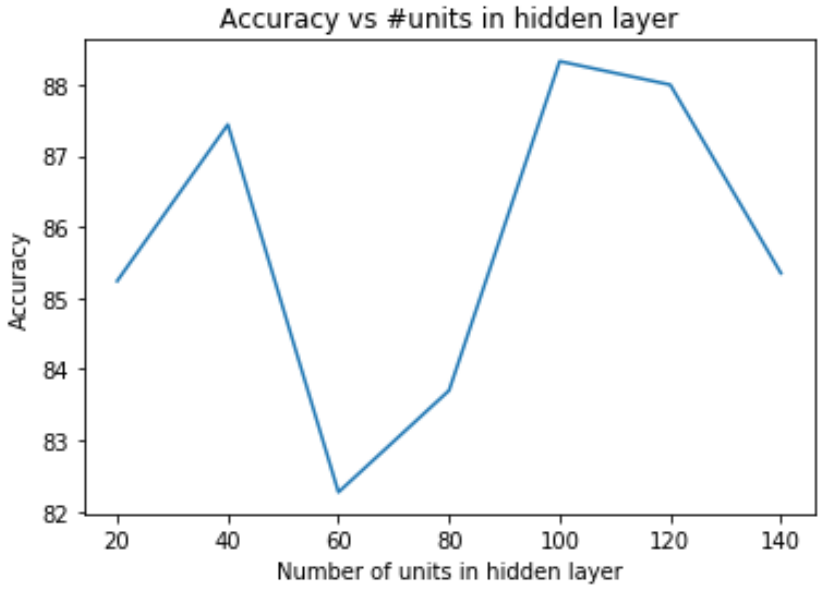


Figure 22: Accuracy vs Hidden Units in Hidden layer

3.4.2 SVM

In our problem, a radial basis function (RBF) kernel is used to train the SVM because such a kernel would be required to address this nonlinear problem.

Large value of parameter C should cause a small margin, and the opposite. Small value of parameter C should cause a large margin. There is no rule to choose a C value, it totally depends on our testing data. The only way is to try different values of the parameters and go with those that gives the highest classification accuracy on the testing phase. As for the gamma parameter, if its value is low then even the far away points can be taken into consideration when drawing the decision boundary and the opposite. With a high gamma parameter, the hyperplane is dependent on the very close points and ignores the ones that are far away from it. So, the best parameters we used in the SVM technique, after testing several variants with trial-and-error approach, were 10 for the C value, and 1 for the gamma value.

3.4.3 RF

The parameters we used after trying several variants (check the figure below), were 5 estimators - the number of trees in the forest – and 5 as the depth of the tree. The algorithm provided us with a high accuracy result in a short time, specifically 99.01% in 0.01 seconds. Generally, our opinion was that the algorithm provided us with the best results in regards to how simple it was to develop and fast to execute.

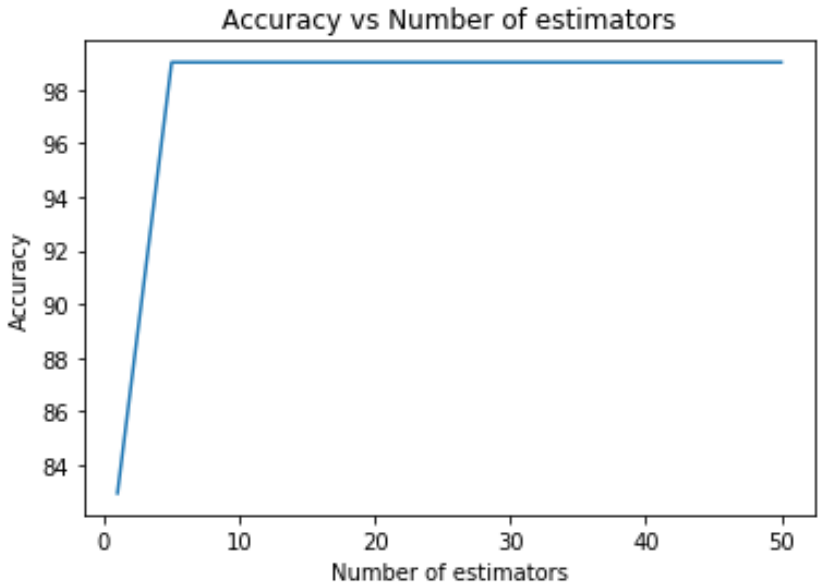


Figure 23: Accuracy vs Number of estimators (RF)

We can use the RF algorithm for feature importance implemented in scikit-learn. Feature importance refers to a class of techniques for assigning scores to input features to a predictive

model that indicates the relative importance of each feature when making a prediction. After being fit, the model provides a feature importance property that can be accessed to retrieve the relative importance scores for each input feature.

In our model, the first 6 features are the most important and we will achieve relatively great accuracy results even if we do not take into account the rest of the features' set. We can see this in the figure below, where the bar chart has bigger values for these features.

- Number of ports: the number of ports in a circuit. In electrical circuit theory, a port is a pair of terminals connecting an electrical network or circuit to an external circuit, a point of entry or exit for electrical energy.

- Number of combinational cells: combinational cells are made up from basic logic NAND, NOR or NOT gates that are "combined" or connected together to produce more complicated switching circuits.

- Number of buf/inv: number of inverter and buffer cells instantiated in the design

- Buf/inv area: area occupied by the above

- Total Cell Area

- Total Dynamic Power: Total Power consumed by the design, given switching activity produced by vectors with coverage > 95%

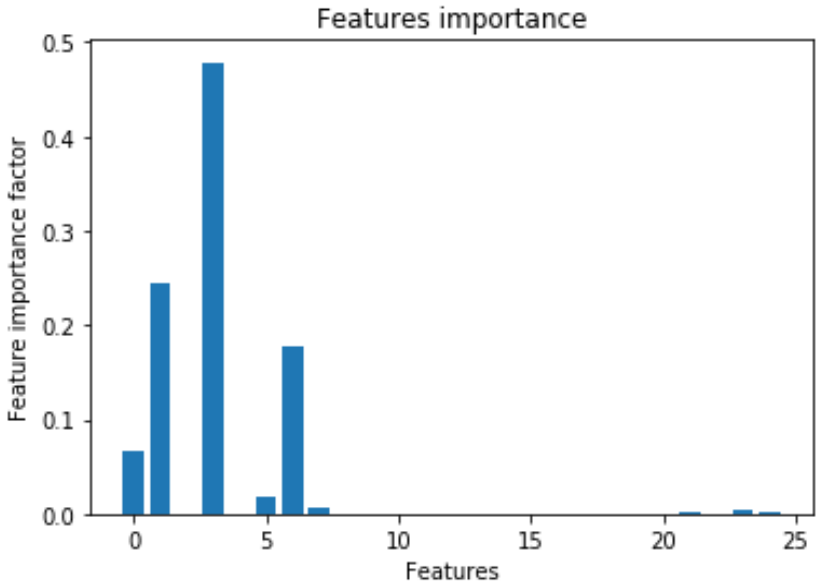


Figure 24: Features importance graph

3.4.4 GB

After trying several variants, we set the values 0.1 and 75 (with the smaller execution time) to the learning rate and the estimators - the number of boosting stages to perform – respectively and train our model with 99.56% accuracy.

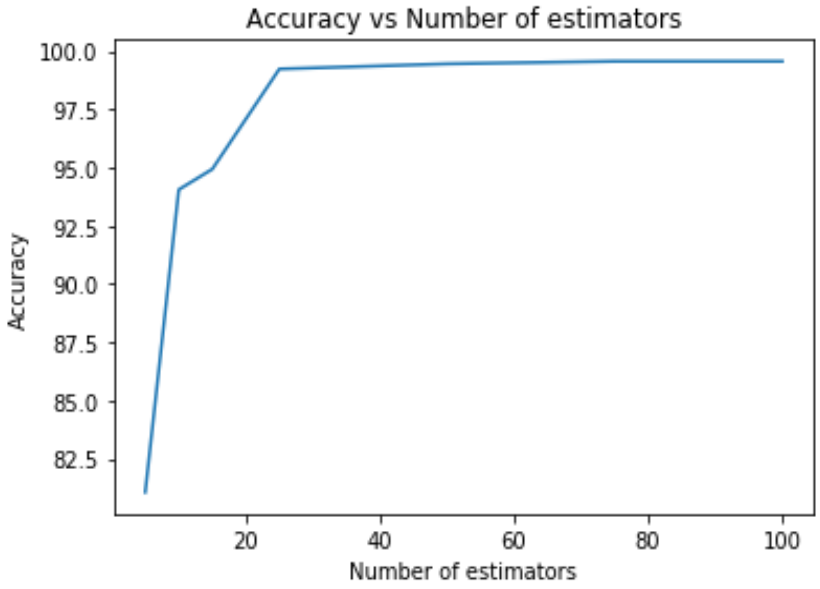


Figure 25: Accuracy vs Number of estimators (GB)

3.4.5 KNN

In our problem, we choose 3 as the number of neighbors, despite of the fact that we achieve the highest accuracy with 1, since we have better execution time.

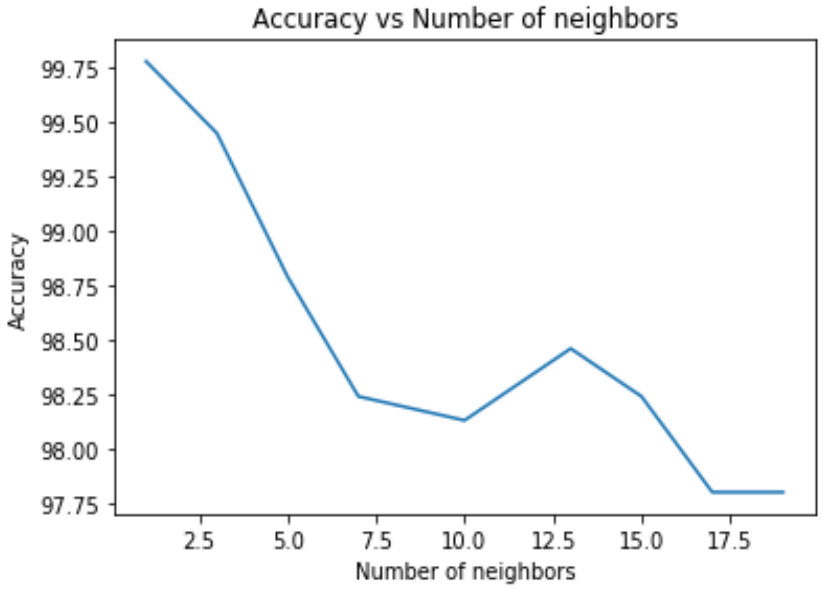


Figure 26: Accuracy vs Number of neighbors

3.4.6 LR

In our dataset, we use binary LR since we have 2 discrete classes for our predicted labels. We use Sklearn's 'LogisticRegression' classifier class, 'liblinear' solver since we have a small dataset and 'or' since the data is binary. In order for the method to converge we choose 300 as max iterations number.

3.4.7 XGB

In our problem, we choose the number of 20 estimators as it provides the highest accuracy in the shortest execution time.

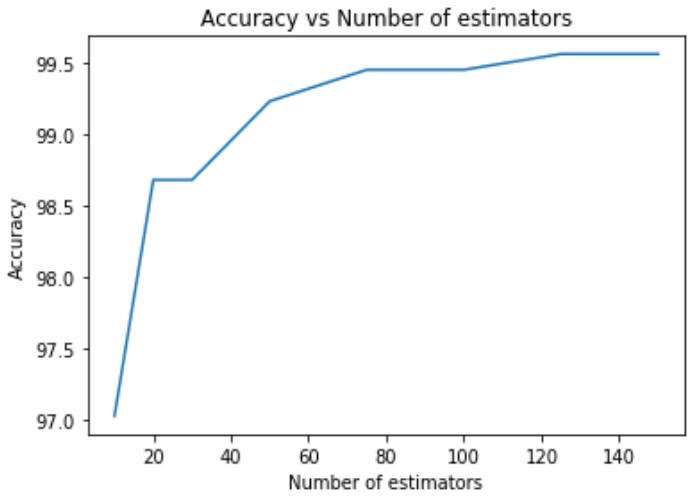


Figure 27: Accuracy vs Number of estimators (XGB)

Chapter 4. Comparison of the Results

4.1 Comparing the Results for our dataset

Our baseline was single-threaded CPU code. The libraries in Python and Sklearn are commonly used for ML purposes, so they are the de facto standard. The characteristics of our system include an Intel Core i7-4510U CPU @ 2.00GHz x4 processor, 8 GB Ram, 4 cores and OS type of Ubuntu 19.04.

4.1.1 Model comparison on our testing-set

One way of visualizing the performance of our models is through a confusion matrix. A confusion matrix is a performance measurement for ML classification problem where output can be two or more classes. For a binary classification problem, it is a table with 4 different combinations of predicted and actual values.

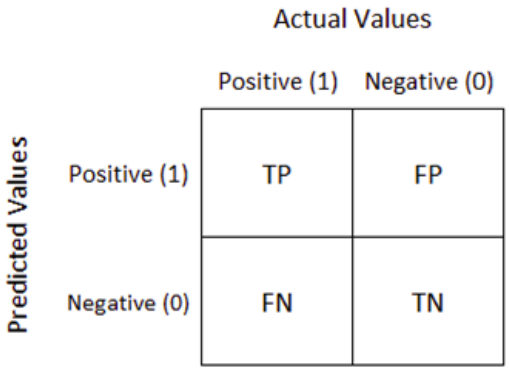


Figure 28: Confusion Matrix of binary classification problem

- True positives (TP): In our problem of predicting whether an instance is TF (0) or TI (1), these are cases in which we predicted TF and it is also a TI instance.
- True negatives (TN): We predicted TI, and it is TI instance.
- False positives (FP): We predicted TF, but it is TI.
- False negatives (FN): We predicted TI, but it is actually a TF one.

Also, we took into consideration various metrics in order to choose the best algorithm for our problem: Precision, Sensitivity-Recall, Specificity, F-measure.

Metric	Description	Type
Precision	the proportion of positive results that truly are positive.	$\frac{TP}{TP + FP}$
Sensitivity - Recall	the proportion of actual positives that are correctly identified as such	$\frac{TP}{TP + FN}$
Specificity	the proportion of actual negatives that are correctly identified as such	$\frac{TN}{TN + FP}$
F-measure	a test's accuracy that balances the use of precision and recall to do it	$\frac{2 * Precision * Recall}{Precision + Recall}$

Table 3: Quantitative Metrics

Below, we provide the comparison between the used algorithms in order to give a better look into which algorithm is considered to be more suitable when we test our models on the testing set.

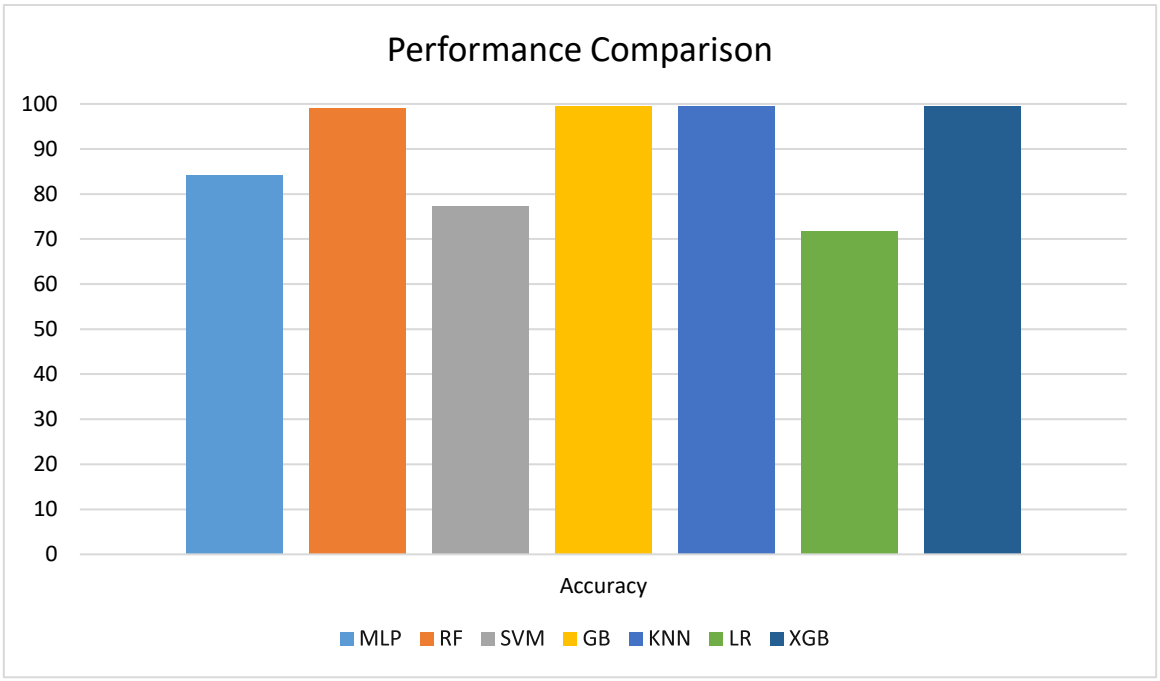


Figure 29: Accuracy comparison of the algorithms (testing-set)

ALGORITHM	ACCURACY	EXECUTION TIME
MLP	84.25 %	4.458 sec
RF	99 %	0.013 sec
SVM	77.3 %	0.217 sec
GB	99.56 %	0.142 sec
KNN	99.45 %	0.084 sec
LR	71.81 %	0.004 sec
XGB	99.45 %	2.706 sec

Table 4: Model comparison for testing on the testing-set

Figure 30: Metrics comparison of the algorithms (testing-set)

Here, as seen in the figure below GB algorithm did well with classifying the most samples into the correct class. A small percentage was predicted as TI but it was TF in reality.

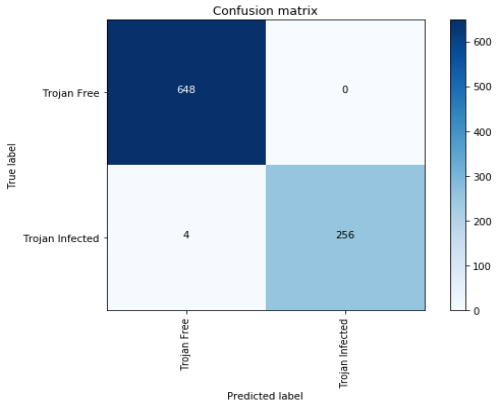


Figure 31: Confusion Matrix of our Testset

From the above-mentioned metrics and visualizations, it is obvious that GB produces the better results when performed on our training-set.

4.1.2 Model comparison on our training-set

Below, we provide a performance comparison between the used algorithms in order to give a better look into which algorithm is considered to be more suitable when we test our models on the training set.

Figure 32: Accuracy comparison of the algorithms (training-set)

ALGORITHM	ACCURACY	EXECUTION TIME	F-measure
MLP	86.11 %	2.209 sec	88.28%
RF	99.29 %	0.009 sec	99.15%
SVM	77.29 %	0.199 sec	63.11%
GB	**99.95 %**	**0.142 sec**	**99.94%**
KNN	99.67 %	0.045 sec	99.61%
LR	71.62 %	0.004 sec	45.86%
XGB	99.62 %	0.304 sec	99.54%

Table 5: Model comparison for testing on the training-set

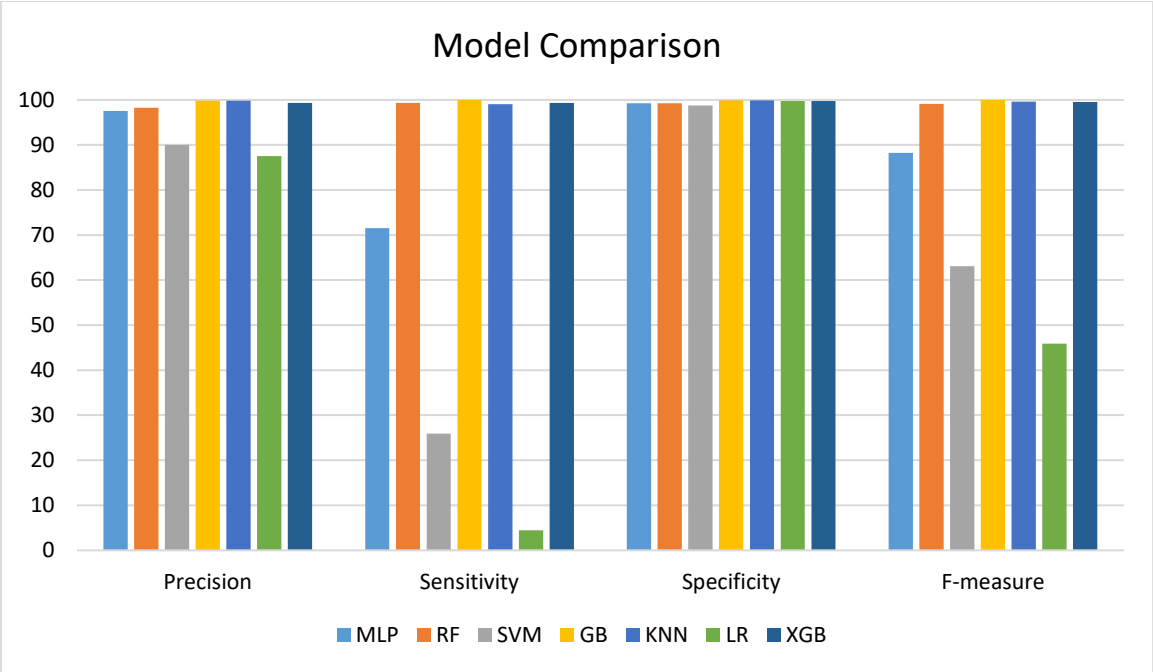


Figure 33: Metrics comparison of the algorithms (training-set)

It is obvious that GB produces the better results - on the training set - combining the above-mentioned metrics and visualizations.

4.2 Comparing the Results to the existing work

M. Xue [65] had trained 50 TI and 50 TF instances and achieved 98% accuracy on unknown Trojan-inserted ICs. Also, achieved 100% accuracy on known Trojan-inserted ICs. In both cases they made use of SVM algorithm on the S38417 and S35932 circuit categories. Based on the same circuit categories, we trained 198 TF and 66 TI instances and achieved the following results. More specifically, using GB we hit 100% accuracy on unknown Trojan-inserted ICs and 100% accuracy on known Trojan-inserted ICs.

H. Salmani [66] had trained an unknown of TI and TF instances and achieved 100% accuracy (0 FN, 0 FP). He made use of K-Means algorithm on various circuit categories. Based on the same circuit categories, we achieved 99.71% accuracy (0 FP, 1 FN) and 99.54% F-measure using GB algorithm.

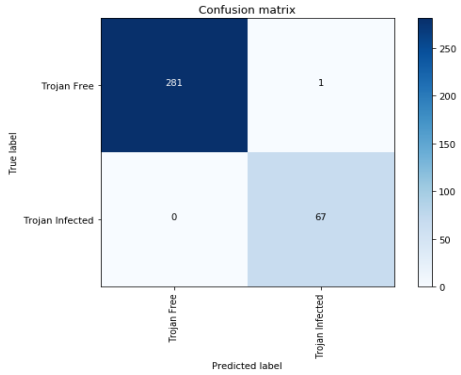


Figure 34: Confusion matrix on dataset of paper [66]

K. Hasegawa [67] had achieved 80% - 100% TPR using SVM algorithm. TPR metric is calculated as: TPR = TP / (TP+FP). In our case, we achieved 98% TPR since we had 60 TP and 1 FN.

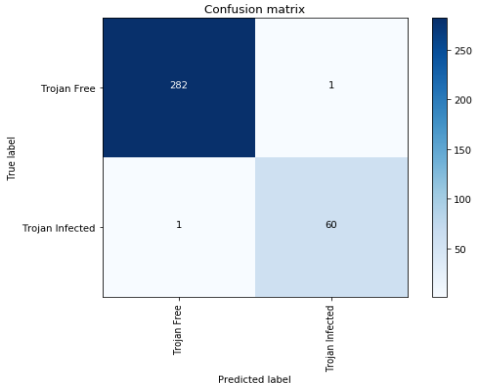


Figure 35: Confusion matrix on dataset of paper [67]

In another study K. Hasegawa [68] had achieved 74.6% F-measure using RF algorithm. In our case, we achieved 99.50% F-measure using GB algorithm.

Below we provide the comparison results of our best algorithm (GB) with the performance of the previous 4 papers mentioned above.

Author	ALGORITHM	ACCURACY	F-measure
[65]	SVM	98%	-
CasLab-HT	**GB**	**100 %**	**100%**
[66]	K-Means	100%	-
CasLab-HT	**GB**	**99,71%**	**99.54%**
[67]	SVM	80-100%	-
CasLab-HT	**GB**	**99.71%**	**99.5%**
[68]	RF	-	74.6%
CasLab-HT	**GB**	**99.71%**	**99.50%**

Table 6: Comparison results with the existing research papers

Chapter 5. Conclusion

Every year, the development rate as well as the sophistication level of HTs are increasing at an alarming rate. Highly elaborate strategies are constantly being introduced for designing HTs that can affect a wide range of ICs, spanning from simple configurations used in most of our everyday life habits to state-of-the-art circuits fabricated for military, industrial and financial purposes or even for state-sponsored sensitive research areas. HTs are stealthy by nature, they come in different form, size and type flavors, while each year HT attacks cost the global economy and mainly the technology companies millions of dollars. If left unchecked, these intrusion attempts will become the most significant obstacle of technological progress in the future, with a severe impact on all aspects of our everyday lives. In this landscape of increased uncertainty related to secure IC design and manufacturing, the need for highly effective, versatile and scalable HT countermeasure methods naturally emerges.

In this thesis in order to identify the HTs, we proposed an HT detection and classification technique via area and power features for GLN phase on ASIC circuits. At first, we design with the Design Compiler NXT software, all the circuits from Trust-HUB library. After that, we developed and compared seven ML models, MLP, RF, SVM, GB, KNN, LR and XGB. We choose the GB model which returned the highest accuracy and F-measure, 100%. And finally, we compared our model with existed state of the art models from journal and conference studies. Our model returned the highest results with average 99.7% accuracy and 99.5 F-measure.

Chapter 6. Future Work

Future work will include the following improvements. The first one is the ability of our model implementation to classify each sample into a circuit category. In that way, we could find out on which circuit category our methods failed to analyze its features or/and leak information from their values. This can be done by setting the ‘circuit category’ feature as the target value, taking into consideration any subcategories and grouping them to the main ones (eg. 'S38584-T100', 'S38584-T200' and 'S38584-T300' to 'S38584’ team).

The second would make our problem able to deal with real – time data, so we could create our dataset’s samples by inserting HT attacks into an IC. This could be done by having access to this circuit and analyzing the stream of data we passed to it and taking advantage of the already known output and comparing to the real output that the circuit produces. An instant to way to see how a HT can affect the structure of an IC.

Last but not least, a simple improvement could be to expand the number of samples that our datasets contain, while keeping the balance between the number of Hardware Free and Hardware Infected samples. We would have then a better and more representative prediction results.

REFERENCES

[1] S. Bhunia *et al.*, “Protection against hardware trojan attacks: Towards a comprehensive solution,” *IEEE Des. Test*, 2013, doi: 10.1109/MDT.2012.2196252.

[2] K. G. Liakos, G. K. Georgakilas, S. Moustakidis, N. Sklavos, and F. C. Plessas, “Conventional and Machine Learning Approaches as Countermeasures against Hardware Trojan Attacks :”

[3] C. Rooney, A. Seeam, and X. Bellekens, “Creation and detection of hardware trojans using non-invasive off-the-shelf technologies,” *Electron.*, 2018, doi: 10.3390/electronics7070124.

[4] Y. Jin and Y. Makris, “Hardware Trojan detection using path delay fingerprint,” in *2008 IEEE International Workshop on Hardware-Oriented Security and Trust, HOST*, 2008, doi: 10.1109/HST.2008.4559049.

[5] M. Potkonjak, A. Nahapetian, M. Nelson, and T. Massey, “Hardware trojan horse detection using gate-level characterization,” in *Proceedings - Design Automation Conference*, 2009, doi: 10.1145/1629911.1630091.

[6] M. Banga and M. S. Hsiao, “A novel sustained vector technique for the detection of hardware trojans,” in *Proceedings: 22nd International Conference on VLSI Design - Held Jointly with 7th International Conference on Embedded Systems*, 2009, doi: 10.1109/VLSI.Design.2009.22.

[7] M. S. Anderson, C. J. G. North, and K. K. Yiu, “Towards Countering the Rise of the Silicon Trojan,” 2008.

[8] M. Abramovici and P. Bradley, “Integrated circuit security - New threats and solutions,” in *ACM International Conference Proceeding Series*, 2009, doi: 10.1145/1558607.1558671.

[9] L. A. Guimarães, “Testing Techniques for Detection of Hardware Trojans in Integrated Circuits of Trusted Systems.,” *Micro and nanotechnologies/Microelectronics*, 2018, [Online]. Available: https://tel.archives-ouvertes.fr/tel-01754790/document.

[10] K. G. Liakos, G. K. Georgakilas, S. Moustakidis, P. Karlsson, and F. C. Plessas, “Machine Learning for Hardware Trojan Detection: A Review,” in *5th Panhellenic Conference on Electronics and Telecommunications, PACET 2019*, Nov. 2019, doi: 10.1109/PACET48583.2019.8956251.

[11] S. B. Kotsiantis, “Supervised machine learning: A review of classification techniques,” *Informatica (Ljubljana)*. 2007, doi: 10.31449/inf.v31i3.148.

[12] S. M. Chelly and C. Denis, “Applying Unsupervised Learning,” *Med. Sci. Sports Exerc.*, 2001, doi: 10.1111/j.2041-210X.2010.00056.x.

[13] K. Pearson, “ LIII. On lines and planes of closest fit to systems of points in space ,” *London, Edinburgh, Dublin Philos. Mag. J. Sci.*, vol. 2, no. 11, pp. 559–572, Nov. 1901, doi: 10.1080/14786440109462720.

[14] R. Vrasti *et al.*, “Reliability of the Romanian version of the alcohol module of the WHO Alcohol Use Disorder and Associated Disabilities: Interview Schedule - Alcohol/Drug-Revised,” *Eur. Addict. Res.*, vol. 4, no. 4, pp. 144–149, 1998, doi: 10.1159/000018947.

[15] R. A. FISHER, “THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS,” *Ann. Eugen.*, vol. 7, no. 2, pp. 179–188, Sep. 1936, doi: 10.1111/j.1469-1809.1936.tb02137.x.

[16] D. J. Olive, *Linear regression*. 2017.

[17] D. R. Cox, “The Regression Analysis of Binary Sequences,” *J. R. Stat. Soc. Ser. B*, vol. 20,

no. 2, pp. 215–232, Jul. 1958, doi: 10.1111/j.2517-6161.1958.tb00292.x.
[18] M. G. Hossain, R. Zyroul, B. P. Pereira, and T. Kamarul, “Multiple regression analysis of factors influencing dominant hand grip strength in an adult Malaysian population,” *J. Hand Surg. (European Vol.*, vol. 37, no. 1, pp. 65–70, Jan. 2012, doi: 10.1177/1753193411414639.
[19] Ö. İ. Güneri and A. Göktaş, “A Comparion of Ordinary Least Squares Regression and Least Squares Ratio via Generated Data,” *Am. J. Math. Stat.*, vol. 7, no. 2, pp. 60–70, 2017, doi: 10.5923/j.ajms.20170702.02.
[20] J. H. Friedman, “Multivariate Adaptive Regression Splines,” *Ann. Stat.*, vol. 19, no. 1, pp. 1–67, 1991, doi: 10.1214/AOS/1176347963.
[21] J. R. Quinlan and J. R. Quinlan, “Learning With Continuous Classes,” pp. 343--348, 1992, Accessed: Jun. 12, 2020. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.34.885.
[22] W. S. Cleveland, “Robust locally weighted regression and smoothing scatterplots,” *J. Am. Stat. Assoc.*, vol. 74, no. 368, pp. 829–836, 1979, doi: 10.1080/01621459.1979.10481038.
[23] R. C. Tryon, “Communality of a variable: Formulation by cluster analysis,” *Psychometrika*, vol. 22, no. 3, pp. 241–260, Sep. 1957, doi: 10.1007/BF02289125.
[24] S. P. Lloyd, “Least Squares Quantization in PCM,” *IEEE Trans. Inf. Theory*, vol. 28, no. 2, pp. 129–137, 1982, doi: 10.1109/TIT.1982.1056489.
[25] S. C. Johnson, “Hierarchical clustering schemes,” *Psychometrika*, vol. 32, no. 3, pp. 241–254, Sep. 1967, doi: 10.1007/BF02289588.
[26] H. Baron, “Bovine spongiform encephalopathy and related agents: Risk factors of biological products. An informal consultation/working meeting,” *Toxicol. Pathol.*, vol. 19, no. 3, pp. 293–297, 1991, doi: 10.1177/019262339101900314.
[27] S. J. Russell and P. Norvig, “Artificial intelligence - a modern approach: the intelligent agent book,” 1995. Accessed: Jun. 13, 2020. [Online]. Available: http://bit.ly/1pZDFd2http://goo.gl/RTIhkhttp://www.barnesandnoble.com/s/?store=book&keyword=Artificial+Intelligence%3A+A+Modern+Approach+%3B+%5Bthe+Intelligent+Agent+Book%5D.
[28] H. E. Kyburg and J. Pearl, “Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.,” *J. Philos.*, vol. 88, no. 8, p. 434, Aug. 1991, doi: 10.2307/2026705.
[29] U. Grenander, R. O. Duda, and P. E. Hart, “Pattern Classification and Scene Analysis.,” *J. Am. Stat. Assoc.*, vol. 69, no. 347, p. 829, Sep. 1974, doi: 10.2307/2286028.
[30] R. E. Neapolitan, “Models for reasoning under uncertainty,” *Appl. Artif. Intell.*, vol. 1, no. 4, pp. 337–366, 1987, doi: 10.1080/08839518708927979.
[31] E. Fix and J. L. Hodges, “Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties,” *Int. Stat. Rev. / Rev. Int. Stat.*, vol. 57, no. 3, p. 238, Dec. 1989, doi: 10.2307/1403797.
[32] C. G. Atkeson, A. W. Moore, and S. Schaal, “Locally Weighted Learning,” *Artif. Intell. Rev.*, vol. 11, no. 1/5, pp. 11–73, 1997, doi: 10.1023/A:1006559212014.
[33] T. Kohonen, “Learning vector quantization,” *Neural Networks*, vol. 1, no. 1 SUPPL, p. 303, Jan. 1988, doi: 10.1016/0893-6080(88)90334-6.
[34] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*. CRC Press, 2017.
[35] S. L. Salzberg, “C4.5: Programs for Machine Learning by J. Ross Quinlan. Morgan Kaufmann Publishers, Inc., 1993,” *Mach. Learn.*, vol. 16, no. 3, pp. 235–240, Sep. 1994,

doi: 10.1007/bf00993309.
[36] G. V. Kass, “An Exploratory Technique for Investigating Large Quantities of Categorical Data,” *Appl. Stat.*, vol. 29, no. 2, p. 119, 1980, doi: 10.2307/2986296.
[37] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *Bull. Math. Biophys.*, vol. 5, no. 4, pp. 115–133, Dec. 1943, doi: 10.1007/BF02478259.
[38] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain,” *Psychol. Rev.*, vol. 65, no. 6, pp. 386–408, Nov. 1958, doi: 10.1037/h0042519.
[39] D. S. Broomhead and D. Lowe, “Multivariable Functional Interpolation and Adaptive Networks,” *undefined*, 1988.
[40] S. Linnainmaa, “Taylor expansion of the accumulated rounding error,” *BIT*, vol. 16, no. 2, pp. 146–160, 1976, doi: 10.1007/BF01931367.
[41] M. Riedmiller and H. Braun, “Direct adaptive method for faster backpropagation learning: The RPROP algorithm,” in *1993 IEEE International Conference on Neural Networks*, 1993, pp. 586–591, doi: 10.1109/icnn.1993.298623.
[42] S. K. Pal and S. Mitra, “Multilayer Perceptron, Fuzzy Sets, and Classification,” *IEEE Trans. Neural Networks*, vol. 3, no. 5, pp. 683–697, 1992, doi: 10.1109/72.159058.
[43] R. Hecht-Nielsen, “Counterpropagation networks,” *Appl. Opt.*, vol. 26, no. 23, p. 4979, Dec. 1987, doi: 10.1364/AO.26.004979.
[44] J. S. R. Jang, “ANFIS: Adaptive-Network-Based Fuzzy Inference System,” *IEEE Trans. Syst. Man Cybern.*, vol. 23, no. 3, pp. 665–685, 1993, doi: 10.1109/21.256541.
[45] W. Melssen, R. Wehrens, and L. Buydens, “Supervised Kohonen networks for classification problems,” *Chemom. Intell. Lab. Syst.*, vol. 83, no. 2, pp. 99–113, Sep. 2006, doi: 10.1016/j.chemolab.2006.02.003.
[46] J. J. Hopfield, “Neural networks and physical systems with emergent collective computational abilities.,” *Proc. Natl. Acad. Sci. U. S. A.*, vol. 79, no. 8, pp. 2554–2558, Apr. 1982, doi: 10.1073/pnas.79.8.2554.
[47] G. Bin Huang, Q. Y. Zhu, and C. K. Siew, “Extreme learning machine: Theory and applications,” *Neurocomputing*, vol. 70, no. 1–3, pp. 489–501, Dec. 2006, doi: 10.1016/j.neucom.2005.12.126.
[48] T. Kohonen, “The Self-Organizing Map,” *Proc. IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990, doi: 10.1109/5.58325.
[49] D. F. Specht, “A General Regression Neural Network,” *IEEE Trans. Neural Networks*, vol. 2, no. 6, pp. 568–576, 1991, doi: 10.1109/72.97934.
[50] J. Cao, Z. Lin, and G. Bin Huang, “Self-adaptive evolutionary extreme learning machine,” *Neural Process. Lett.*, vol. 36, no. 3, pp. 285–305, Dec. 2012, doi: 10.1007/s11063-012-9236-y.
[51] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553. Nature Publishing Group, pp. 436–444, May 27, 2015, doi: 10.1038/nature14539.
[52] I. Goodfellow, Y. Bengio, and A. Courville, “RegGoodfellow, I., Bengio, Y., & Courville, A. (2016). Regularization for Deep Learning. Deep Learning, 216–261.ularization for Deep Learning,” *Deep Learn.*, 2016.
[53] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. A. Manzagol, “Stacked denoising autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion,” *J. Mach. Learn. Res.*, 2010.
[54] R. Salakhutdinov and G. Hinton, “Deep Boltzmann machines,” in *Journal of Machine*

Learning Research, 2009.
[55] C. Cortes and V. Vapnik, “Support-Vector Networks,” *Mach. Learn.*, 1995, doi: 10.1023/A:1022627411411.
[56] J. A. K. Suykens and J. Vandewalle, “Least squares support vector machine classifiers,” *Neural Process. Lett.*, 1999, doi: 10.1023/A:1018628609742.
[57] C. C. Chang and C. J. Lin, “LIBSVM: A Library for support vector machines,” *ACM Trans. Intell. Syst. Technol.*, 2011, doi: 10.1145/1961189.1961199.
[58] A. J. Smola and B. Schölkopf, “A tutorial on support vector regression,” *Statistics and Computing*. 2004, doi: 10.1023/B:STCO.0000035301.49549.88.
[59] J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle, “Basic Methods of Least Squares Support Vector Machines,” in *Least Squares Support Vector Machines*, 2002.
[60] R. K. H. Galvão *et al.*, “A variable elimination method to improve the parsimony of MLR models using the successive projections algorithm,” *Chemom. Intell. Lab. Syst.*, 2008, doi: 10.1016/j.chemolab.2007.12.004.
[61] L. Breiman, “Random forests,” *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001, doi: 10.1023/A:1010933404324.
[62] Y. Freund and R. E. Schapire, “Experiments with a New Boosting Algorithm,” *Proc. 13th Int. Conf. Mach. Learn.*, 1996, doi: 10.1.1.133.1040.
[63] R. E. Schapire, T. Labs, P. Avenue, A. Room, and F. Park, “A Brief Introduction to Boosting Generalization error,” 1999. doi: citeulike-article-id:2373464.
[64] L. Breiman, “Bagging predictors,” *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996, doi: 10.1007/bf00058655.
[65] M. Xue, J. Wang, and A. Hux, “An enhanced classification-based golden chips-free hardware Trojan detection technique,” in *Proceedings of the 2016 IEEE Asian Hardware Oriented Security and Trust Symposium, AsianHOST 2016*, 2017, doi: 10.1109/AsianHOST.2016.7835553.
[66] H. Salmani, “COTD: Reference-Free Hardware Trojan Detection and Recovery Based on Controllability and Observability in Gate-Level Netlist,” *IEEE Trans. Inf. Forensics Secur.*, 2017, doi: 10.1109/TIFS.2016.2613842.
[67] K. Hasegawa, M. Oya, M. Yanagisawa, and N. Togawa, “Hardware Trojans classification for gate-level netlists based on machine learning,” in *2016 IEEE 22nd International Symposium on On-Line Testing and Robust System Design, IOLTS 2016*, 2016, doi: 10.1109/IOLTS.2016.7604700.
[68] K. Hasegawa, M. Yanagisawa, and N. Togawa, “Trojan-feature extraction at gate-level netlists and its application to hardware-Trojan detection using random forest classifier,” in *Proceedings - IEEE International Symposium on Circuits and Systems*, 2017, doi: 10.1109/ISCAS.2017.8050827.

APPENDIX

The python code below was used to fit our models. We consider Sklearn as the basic package for our models and our implementation as well as with the parameters of each function can be seen below:

MLP

```
model = Sequential()
model.add(Dense(15, input_dim=train_x.shape[1], activation='relu'))
model.add(Dense(75, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['mse', 'accuracy'])
model.fit(train_x, train_y, epochs=50, batch_size=150, shuffle=False, verbose=0)
```

Figure 36: MLP model using Keras, Tensorflow high-level API

SVM

```
C = 10
gamma = 1
classifier = SVC(kernel="rbf",C=C,gamma=gamma)
classifier.fit(train_x, train_y)
y_pred = classifier.predict(test_x)
```

Figure 37: SVM using Sklearns SVC classifier class

RF

```
clf = RandomForestClassifier(n_estimators=5, max_depth=5, random_state=1)
clf.fit(train_x, train_y)
y_pred = clf.predict(test_x)
```

Figure 38: RF using Sklearns RF classifier class

GB

```
clf = GradientBoostingClassifier(learning_rate=0.1, n_estimators=75)
clf.fit(train_x, train_y)
y_pred = clf.predict(test_x)
```

Figure 39: GB using Sklearns GradientBoostingClassifier class

KNN

```
clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(train_x, train_y)
y_pred = clf.predict(test_x)
```

Figure 40: KNN using Sklearns KNeighborsClassifier class

LR

```
clf = LogisticRegression(random_state=0, solver='liblinear', max_iter=300,
                         multi_class='ovr')
clf.fit(train_x, train_y)
y_pred = clf.predict(test_x)
```

Figure 41: LR using Sklearns LogisticRegression classifier class

XGB

```
clf = XGBClassifier(n_estimators=20)
clf.fit(train_x, train_y)
y_pred = clf.predict(test_x)
```

Figure 42: XGB using Sklearns XGBClassifier class