Informatics Engineering Graduation

# Applying Machine Learning Techniques to Improve Intrusion Detection

a project authored by

Pedro Dias & Rui Freitas

supervised by

Prof. Pedro Pinto and Prof. Silvestre Malta

29 June, 2020

**Abstract**

Nowadays, the rate of attacks or attack attempts on networks and machines, easily reaches several thousands each day. With the evolution of hacking tools and techniques, making them easier to handle, apply and/or deploy, one can use a machine to sweep a wide range of networks and other machines for vulnerabilities. Without much effort, this way, attackers can get list of vulnerable machines and their specific vulnerabilities, and can even automatically perform previously programmed attacks, without having to actively perform all the attack tasks and having only to collect the results. This automatic attacking sweep techniques, and the associated attack types, can be dealt with Intrusion Detection Systems, which need to be aware of the existing attack types and their *modus operandi*. In this article machine learning techniques are applied for the detection of several types of attacks. This is achieved with the analysis of previously attack type classified network flows datasets, the creation of a prediction model and applying it to new network flows. The results are very satisfying, on certain ML models reaching to values around 99% of accuracy.

**Keywords:** Machine Learning. Intrusion Detection System. Intrusion Prevention Sytem. HoneyPot. CICIDS2017. Cybersecurity. Naive Baye. Decision tree. K-nearest neighbors. Logistic Regression.

# Contents

# Acronyms

|  |  |
|---|---|
| **ASM** | Attribute Selection Measures |
| **CIC** | Canada Institute for Cybersecurity |
| **CSV** | Comma-separated Values |
| **DT** | Decision Tree |
| **EFS** | Ensemble Feature Selection |
| **HIDS** | Host Intrusion Detection System |
| **HN** | HoneyPot |
| **ID3** | Iterative Dichotomiser 3 |
| **IDS** | Intrusion Detection System |
| **IP** | Internet Protocol |
| **IPS** | Intrusion Prevention System |
| **KNN** | K-Nearest Neighbors |
| **LR** | Logistic Regression |
| **ML** | Machine Learning |
| **MLE** | Maximum Likelihood Estimation |
| **NB** | Naive Baye |
| **NID** | Network Intrusion Detection |

| | |
|---|---|
| **NIDS** | Network Intrusion Detection System |
| **OLS** | Ordinary Least Squares |
| **PCA** | Principal Component Analysis |
| **SMOTE** | Synthetic Minority Oversampling Technique |
| **VM** | Virtual Machine |
| **WEKA** | Waikato Environment for Knowledge Analysis |
| **WN** | Wireless Network |

# Chapter 1

# Introduction

## 1.1 Context and motivation

This paper was developed in context of the class of Project IV, of the 3rd year of the degree in Computer Engineering from the ”Escola Superior de Tecnologia e Gestão” of the Polytechnic Institute of Viana do Castelo. It is intended not only to apply knowledge and concepts acquired during the degree, namely with regard to ”machine learning” and network security, but also to explore and document algorithms and techniques applied to Network Intrusion Detection (NID). With this investigation we intend to make a contribution for anyone who follows a similar path, whatever the final objective, thus having data that allow them to make the right choice in advance when it comes to the algorithm with the best performance in the classification of security breaches.

## 1.2 Problem

Signature based Intrusion Detection System (IDS) do not detect zero-day attacks. This is their main flaw or weakness. Anomaly based IDS scan do it, but they require an extensive and complex sets of rules on all protocols, which takes quite some resources. A network flow analysis IDS can solve some of this, as it does not focus on any behaviour pattern, but on the meta-data of transmitted and received data. Advantages of using machine learning (by data mining) in this situation: deal with high-dimensional data, after creating the models, its use to classify new data is fast, enables the identification of patterns in unsupervised learning;

## 1.3 Objectives

The main objective is to identify which machine learning algorithm, in the context of analysis and classification of network data flow, has better performance in terms of precision and accuracy. However, in order to achieve this objective, several other have been set whose existence becomes essential and mandatory for the success of the project. As such, the objectives of this work are also:

- to identify and study (sets of) existing traffic flow datasets with identified attacks, their analysis and processing;
- the description of the chosen algorithms and why to chose them
- the exploration of machine learning techniques and technologies;
- creating and obtaining classifying models that can be used by the community (with direct applicability).
- to clarify and document the best performing machine learning algorithm for Network Intrusion Detection (NID) (main goal).

## 1.4 Organization

This paper structure obeys a standard organization of content.

After this introduction, we start by establishing the context background and state of the art, with the definition of relevant concepts such as honeypots and IDSs.

Next is a detailed description of the Machine Learning models used throughout the investigation, which are 4: Logistic Regression (LR), Decision Tree (DT), Naive Bayes (NBs) and K-Nearest Neighbors (KNN). There is also description of some Machine Learning algorithms concepts.

Following, we start by exploring the datasets, listing all those we've analysed and the three most relevant ones are described, having a more detailed and thorough description of the chosen dataset. Next, there is a description of the network flow capture tool used and its relationship with the chosen dataset.

In the Results and Analysis section we start by making a general description of tests structure and how they were performed and what different values we used as parameters.

Afterwards, the results of the several different approaches are shown through tables and graphics, aiding the textual description and interpretation of the results.

And then we the make our conclusions. We summarize the results and their meaning, establishing relations between them and concluding with the overall highest scored model.

# Chapter 2

# State of art

In this chapter it is presented knowledge and articles with work done around the area our project is focused on.

## 2.1 Background

In this section it is presented valuable information needed to understand some of the concepts that this project is based on.

### 2.1.1 HoneyPot

As explained in [20], the definition comes from the world of espionage, where a romantic relationship is used to steal secrets. This is described as setting a 'honeypot'. As in [20], "In computer security terms, a cyber honeypot works in a similar way, baiting a trap for hackers. It's a sacrificial computer system that's intended to attract cyberattacks, like a decoy. It mimics a target for hackers, and uses their intrusion attempts to gain information about cybercriminals and the way they are operating or to distract them from other targets. "A honeypot simulates a real machine, a real computer, with programs and data serving as a decoy for attackers that mimics a real target. A lot of companies around the world use this technique to lure the cybercriminals away of the real machines, protecting this way sensible information that their computer may have such as client credit card information. Once they are fooled to break into the machine , the honeypot itself tracks their moves to serve as learning data to make real servers more secure"[20].

As described in [20], Honeypots are deliberately made vulnerable to be attractive to the attacker as a easy target. This can be achieved by using weak password to brute force or guess and even let common ports open. As in [20] , "A honeypot is not set up to address a specific problem, like a firewall or anti-virus. Instead, it is an information tool that can help you understand existing threats to your business and spot the emergence of new threats. With the intelligence obtained from a honeypot, security efforts can be prioritized and focused". Honeypots can be categorized by who uses them and what their primary goal is.

Research honeypots are mainly implemented by security researchers,militaries and governments.They have a high complexity to serve as studying platform to analyze how the hackers advance in the machine to help organizations or researchers identify possible security failures , learn from this information and implement security improvements to prevent attacks.[20]

As explained [20] , Production honeypots are mostly used by organizations , they are implemented along side the IDS that helps scan malicious attempts.

Honeypot systems can also be classified as:

- Pure honeypots: As in [20],"which are full production systems that do not require any other software. In other words, they are production servers made into honeypots, and they are connected to the rest of the network. They are the most believable but also the riskiest and the most expensive ones."

- High-interaction: As in [20], "honeypots are non-emulated operating systems. They imitate production systems and usually have a lot of services and data. Thus they require a lot of resources to function. Such honeypots are usually run on Virtual Machine (VM) as this allows multiple honeypots to run on a single device. This also makes it easier to sandbox compromised systems, shut them down, and restore them."

- Low-interaction: As in [20],"honeypots emulate only the most 'wanted' system or service. They require fewer resources and are also mostly used on VM. Thus they are less risky and easier to maintain. On the other hand, they are easier for hackers to identify and are better used to detect malware spread by botnets and worms."

### 2.1.2 Intrusion Detection System

As in [21] , "An IDS is a network security technology originally built for detecting vulnerability exploits against a target application or computer. Intrusion Prevention System

(IPS) extended IDS solutions by adding the ability to block threats in addition to detecting them and has become the dominant deployment option for IDS/IPS technologies." An IDS exists only to detect threats as so will take advantage of TAP or SPAN port to analyze a copy of inline traffic as cannot keep up with the amount of traffic in real-time communication path [21].As explained in [21], an IDS was developed with the intention described above because could not perform a detection fast enough to keep up with the network. As in [21], an IDS is a listen only device with the intention of monitor the network traffic and report to an administrator to actions being taken , an IDS cannot take actions by itself and prevent attacks to enter the system that is why an IPS is implemented along side.

Different types of IDS can be assumed namely the following ones:

- As in [15] Network Intrusion Detection System (NIDS) , "It is an independent platform that identifies intrusions by examining network traffic and monitors multiple hosts. Network intrusion detection systems gain access to network traffic by connecting to a network hub, a network switch configured for port mirroring, or a network tap. In a NIDS, sensors are placed at choke points in the network to monitor, often in the demilitarized zone (DMZ) or at network borders. Sensors capture all network traffic and analyze the content of individual packets for malicious traffic. An example of a NIDS is Snort."

- As in [15] Host Intrusion Detection System (HIDS), "It consists of an agent on a host that identifies intrusions by analyzing system calls, application logs, file-system modifications (binaries, password files, capability databases, Access control lists, etc.) and other host activities and state. In a HIDS, sensors usually consist of a software agent. Some application-based IDS are also part of this category. An example of a HIDS is OSSEC. IDS can also be system-specific using custom tools and honeypots. In the case of physical building security, IDS is defined as an alarm system designed to detect unauthorized entry."

In case IDS are distinguished by detection method, the following IDS can be assumed:

- As is [22], "Signature-Based IDS This type of IDS is focused on searching for a "signature," patterns, or a known identity, of an intrusion or specific intrusion event. Most IDS are of this type. It needs regular updates of what signatures or identities are common at the moment to ensure its database of intruders is current. This means signature-based IDS is only as good as how up to date its database is at a given moment. Attackers can get around signature-based IDS by frequently changing small things about how the attack takes place, so the databases cannot keep pace. In addition, it means a completely new attack type may not be picked up at all by signature-based IDS because the signature does not exist in the database. Furthermore, the larger the database becomes, the higher the processing load is for the system to analyze each connection and check it against the database."

- As in [22], "Anomaly-Based IDS In contrast to signature-based IDS, anomaly-based IDS looks for the kinds of unknown attacks signature-based IDS finds hard to detect. Due to the rapid growth in malware and attack types, anomaly-based IDS uses machine learning approaches to compare models of trustworthy behavior with new behavior. As a result, strange- or unusual-looking anomalies or behavior will be flagged. However, previously unknown, but legitimate, behavior can be accidentally flagged as well and depending on the response, this can cause some problems. In addition, anomaly-based IDS assumes network behavior always stays predictable and it can be simple to tell good traffic from bad. But anomaly-based IDS looks at the behavior of traffic, not the payload, and if a network is running on a non-standard configuration, the IDS can have problems figuring out which traffic to flag.

  However, anomaly-based IDS is good for determining when someone is probing or sweeping a network prior to the attack taking place. Even these sweeps or probes create signals in the network the anomaly-based IDS will pick up on. This type of IDS needs to be more distributed across the network, and the machine learning processes need to be guided and trained by an administrator."

## 2.2 State of art

In a 2018 paper [7], the authors compared the ability to detect attacks between a NIDS using a Deep Reinforcement Learning Algorithm and machine learning models based on the algorithms J48, Random Forest, Support Vector Machine and Artificial Neural Networks based algorithms.

In a 2018 paper [16], the authors analysis the effectiveness of [6] dataset to apply in a IDS.

In a 2018 paper [4], the authors emphasizes the always changing tools and techniques of the attackers use so as solution the implementation of Machine Learning (ML) models to detect , classify e predict attacks.

In a 2018 paper [18],the authors produces a reliable dataset that contains benign and seven common attack network flows, which meets real world criteria and is publicly available. Consequently, the paper evaluates the performance of a comprehensive set of network traffic features and machine learning algorithms to indicate the best set of features for detecting the certain attack categories.

In a 2019 paper [3],the authors analyzed the [6] dataset as training dataset to compete with a anomaly-based and signature-based IDS to detect attacks.

In a 2019 paper [23], the authors considers the use of Synthetic Minority Oversampling Technique (SMOTE), Principal Component Analysis (PCA), and Ensemble Feature Selection (EFS) to improve the performance of AdaBoost-based IDS on the latest and challenging CICIDS2017 Dataset ,aimed at constructing an improvement performance intrusion detection approach to handle the imbalance of training data, SMOTE is selected to tackle the problem. Moreover, PCA and EFS are applied as the feature selection to select important attributes from the new dataset.

In a 2019 paper [9],the authors apply ML do detect, classify or predict attacks as traditional rule-based security solutions are vulnerable to advanced attacks due to unpredictable behaviors and unknown vulnerabilities.

In a 2019 paper [24],the authors emphasizes the promising methodology of detecting zero-day intrusions with ML models as the anomaly-based or signature-based IDS are not able to detect it.

In a 2020 paper [19],the authors analyze the different types of datasets available to create ML based IDS.

# Chapter 3

# ML Models for IDS

In this chapter is presented a general overview about machine learning algorithm as also a more in deep description of the select algorithms to be used in this project.

## 3.1 Types of machine learning algorithms

There are different ways an algorithm can model a problem based on its interaction with the experience or environment or whatever we want to call the input data.

There are only a few main learning styles or learning models that an algorithm can have:

- Supervised Learning: As in [2], "Input data is called training data and has a known label or result such as spam/not-spam or a stock price at a time. A model is prepared through a training process in which it is required to make predictions and is corrected when those predictions are wrong. The training process continues until the model achieves a desired level of accuracy on the training data. Example problems are classification and regression. Example algorithms include: Logistic Regression and the Back Propagation Neural Network."

- Unsupervised Learning: As in [2] ,"Input data is not labeled and does not have a known result. A model is prepared by deducing structures present in the input data. This may be to extract general rules. It may be through a mathematical process to systematically reduce redundancy, or it may be to organize data by similarity. Example problems are clustering, dimensionality reduction and association rule learning." Example algorithms include: the Apriori algorithm and K-Means.

- Semi-Supervised Learning: As in [2], "Input data is a mixture of labeled and unlabelled examples. There is a desired prediction problem but the model must learn the structures to organize the data as well as make predictions. Example problems are classification and regression. Example algorithms are extensions to other flexible methods that make assumptions about how to model the unlabeled data."

## 3.2 Algorithms

Our main objective is quite clear when it comes to the type of ML problem we want to solve: classification. There's is a set of classifiers/algorithms that are best suited for network analysis and classification. Since testing all these algorithms was not possible due to lack of human, machine and time resources available for this investigation, it was decided to use four of them. The choice would be affected by analysing the pros and cons and thus getting the most balanced ones. The most commun used classification algorithms, according to several data sources, are:

- LogisticRegression
- K-Nearest Neighbors
- Support Vector Machine
- KernelSVM
- NaiveBayes
- Decision Tree
- Random Forest

After analysing the disadvantages, we came to conclusion that the following would be left out of the investigation:

- Support Vector Machine - despite generating good results, this algorithm is not suitable for larger and noisier datasets, as in our case;
- KernelSVM - not chosen for the same reasons of the its predecessor, since its SVM based;

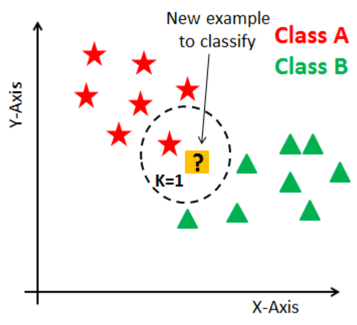


Figure 3.1: First iteration in KNN [11]

- Random Forest - know for its high accuracy over normal decision trees, but since decision trees usually get better results in noisier datasets, in less time/computation, this was also left out.

So the chosen algorithms for the developing of this investigation are the following, which we describe next:

- K-Nearest Neighbors
- Logistic Regression
- Naive Baye
- Decision Tree

### 3.2.1 K-Nearest neighbors

KNN is a non-parametric algorithm , meaning that the structure of the model is determined by the dataset.This way this algorithm can adapt to all of datasets as they don't follow normalized guidelines.KNN is also considered a lazy learning algorithm, this is that all the training data is used in the testing phase , making the training faster and the testing slower and costlier in terms of time and memory used.[10].

As in [11],"In KNN, K is the number of nearest neighbors. The number of neighbors is the core deciding factor. K is generally an odd number if the number of classes is 2. When K=1, then the algorithm is known as the nearest neighbor algorithm. This is the simplest case. Suppose P1 is the point, for which label needs to predict. First,there is a need of find the one closest point to P1 and then the label of the nearest point assigned to P1."

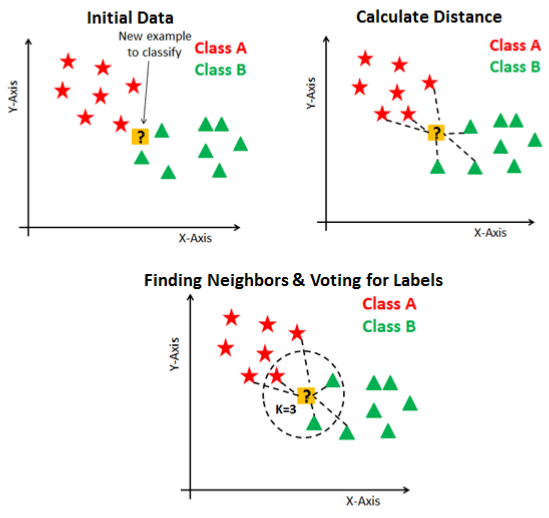


Figure 3.2: Baisc steps of KNN [11]

As describe in [11],"Supposing P1 is the point, for which label needs to predict. First,there is a need to find the k closest point to P1 and then classify points by majority vote of its k neighbors. Each object votes for their class and the class with the most votes is taken as the prediction. For finding closest similar points, you find the distance between points using distance measures such as Euclidean distance, Hamming distance, Manhattan distance and Minkowski distance." [10]

KNN has the following basic steps:

1. Calculate distance
2. Find closest neighbors
3. Vote for labels

The variable K is a parameter that is needed to be choosen at the time of building the model, k will actively impact the model.[11]

As explained in [11],"Research show that there is no optimal number of neighbors to suit all types of data sets, each data set have requirements that need special attention.When the number of neighbors is low the noise will have influence on the result , and when there is a high number of neighbors make it expensive to compute."

Generally, a odd number is chosen if the number of classes is even. It can be checked by generating the model on different values of k and check their performance, as presented in Fig. 3.3.[11]

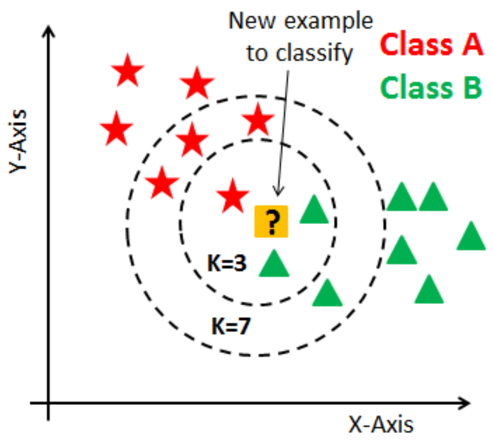


Figure 3.3: Different values of K [11]

### 3.2.2 Logistic Regression

As explained in [12], "Logistic regression is a statistical method for predicting binary classes. The outcome or target variable is dichotomous in nature. Dichotomous means there are only two possible classes.It computes the probability of an event occurrence."

Logistic Regression is a case of linear regression,uses a logic function to predict the probability of a event.

Linear Regression Equation:

$$y = \beta 0 + \beta 1X1 + \beta 2X2 + ... + \beta nXn \tag{3.1}$$

Where, y is dependent variable and x1, x2 ... and Xn are explanatory variables.

Sigmoid Function:

$$p = 1/1 + e^{-y} \tag{3.2}$$

Apply Sigmoid function on linear regression:

$$p = 1/1 + e^{\beta 0+\beta 1X1+\beta 2X2...\beta nXn} \tag{3.3}$$

Properties of Logistic Regression:

- The dependent variable in logistic regression follows Bernoulli Distribution.
- Estimation is done through maximum likelihood.

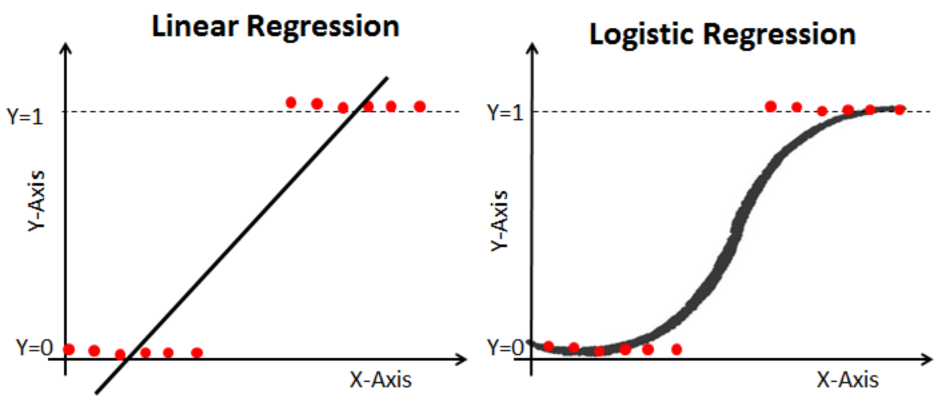


Figure 3.4: Linear Regression vs Logistic Regression

- No R Square, Model fitness is calculated through Concordance, KS-Statistics.

As is explained in [12], "Linear regression gives continuous output, but logistic regression provides a constant output. An example of the continuous output is house price and stock price. An example of discrete output is predicting whether a patient has cancer or not, predicting whether the customer will churn. Linear regression is estimated using Ordinary Least Squares (OLS) while logistic regression is estimated using Maximum Likelihood Estimation (MLE) approach."

As in [12] , The sigmoid function , gives a characteristic output as an 'S' curve 3.5.Has the ability to take any value and assign into a value between 0 and 1.If the curve goes from y=0 to ∞ , y prediction will be 1 , and if the curve goes from y=0 to - ∞, y prediction will be 0.When the outcome is more than 0.5 can be classified as TRUE, and if is less than 0.5 can be classified as FALSE.

$$f(x) = \frac{1}{1 + e^{-(x)}} \tag{3.4}$$

The following types of Logistic Regression can be found in literature:

- Binary Logistic Regression: The target variable has only two possible outcomes such as Spam or Not Spam, Cancer or No Cancer.

- Multinomial Logistic Regression: The target variable has three or more nominal categories such as predicting the type of Wine.

- Ordinal Logistic Regression: the target variable has three or more ordinal categories such as restaurant or product rating from 1 to 5.

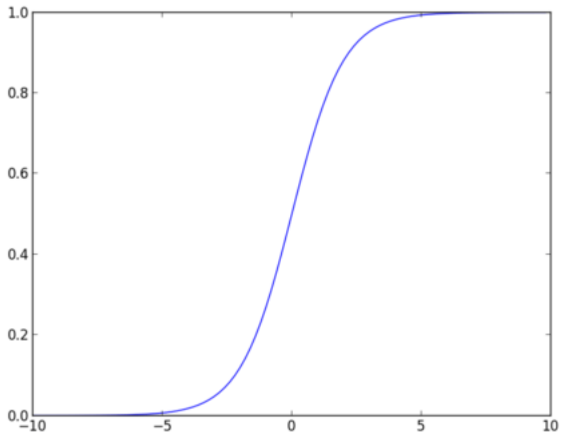


Figure 3.5: sigmoid function

### 3.2.3 Naive Baye

NB is a statistical classification technique based on Bayes Theorem[1]. It is one of the simplest supervised learning algorithms. NB classifier is fast, accurate and reliable algorithm. NBs classifiers have high accuracy and speed on large datasets.

As explained in [13]" NBs classifier assumes that the effect of a particular feature in a class is independent of other features. For example, a loan applicant is desirable or not depending on his/her income, previous loan and transaction history, age, and location. Even if these features are interdependent, these features are still considered independently. This assumption simplifies computation, and that's why it is considered as naive. This assumption is called class conditional independence."

$$\mathrm{P(h|D)} = \frac{\mathrm{P(D|h)P(h)}}{\mathrm{P(D)}} \tag{3.5}$$

- P(h): the probability of hypothesis h being true (regardless of the data). This is known as the prior probability of h.
- P(D): the probability of the data (regardless of the hypothesis). This is known as the prior probability.
- P(h | D): the probability of hypothesis h given the data D. This is known as posterior probability.
- P(D | h): the probability of data d given that the hypothesis h was true. This is known as posterior probability.

### 3.2.4 Decision Tree

As in [5], DT is a flowchart-like tree structure 3.6 where a node represent a feature, the branch represent a decision and each leaf represent an outcome. The first node it is called as the root node.DT works by dividing the tree recursively, this way of deciding is a lot like how humans think or even how if/else works in programming , that is why is easy to understand the way it functions.

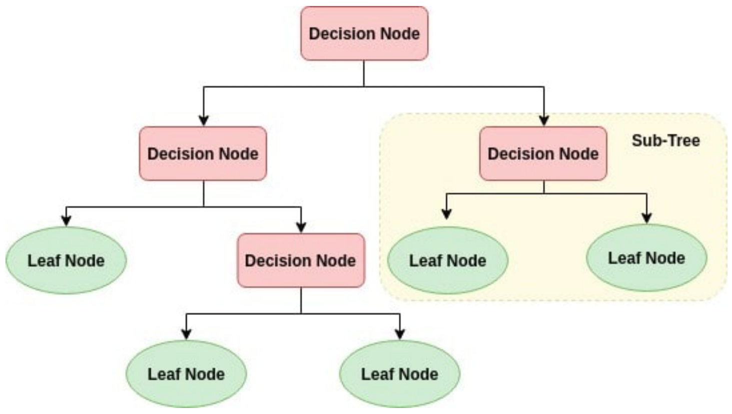


Figure 3.6: Decision tree diagram

As explained in [5], "DT is a white box type of ML algorithm. It shares internal decision-making logic, which is not available in the black box type of algorithms such as Neural Network. Its training time is faster compared to the neural network algorithm. The time complexity of DTs is a function of the number of records and number of attributes in the given data. The DT is a distribution-free or non-parametric method, which does not depend upon probability distribution assumptions. DTs can handle high dimensional data with good accuracy."

The basic idea behind any DT algorithm is as follows:

- Select the best attribute using Attribute Selection Measures (ASM) to split the records.
- Make that attribute a decision node and breaks the dataset into smaller subsets.
- Starts tree building by repeating this process recursively for each child until one of the condition will match:

As in [5], "Attribute selection measure is a heuristic for selecting the splitting criterion that partition data into the best possible manner. It is also known as splitting rules because it helps us to determine breakpoints for tuples on a given node. ASM provides a rank to

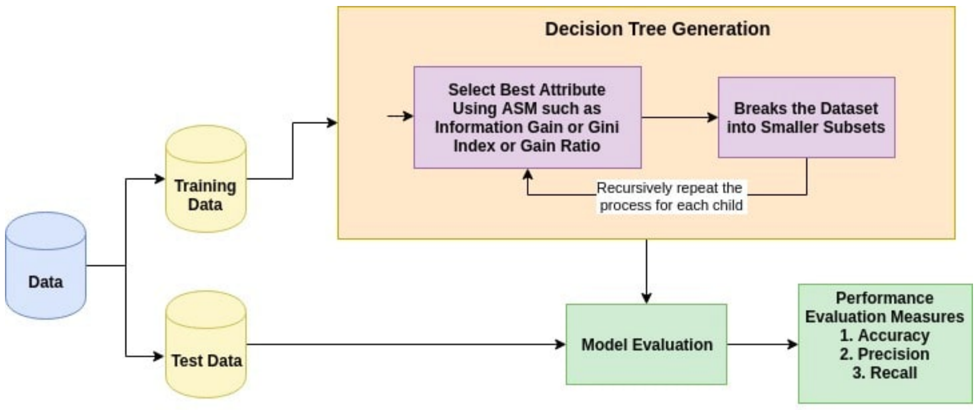

Decision Tree Generation
Select Best Attribute Using ASM such as Information Gain or Gini Index or Gain Ratio
Breaks the Dataset into Smaller Subsets
Recursively repeat the process for each child
Training Data
Data
Test Data
Model Evaluation
Performance Evaluation Measures
1. Accuracy
2. Precision
3. Recall

Figure 3.7: Decision tree flow chart

each feature(or attribute) by explaining the given dataset. Best score attribute will be selected as a splitting attribute (Source). In the case of a continuous-valued attribute, split points for branches also need to define. Most popular selection measures are Information Gain, Gain Ratio, and Gini Index."

As explained in [5], "The concept of entropy, which measures the impurity of the input set. In physics and mathematics, entropy referred as the randomness or the impurity in the system. In information theory, it refers to the impurity in a group of examples. Information gain is the decrease in entropy. Information gain computes the difference between entropy before split and average entropy after split of the dataset based on given attribute values. Iterative Dichotomiser 3 (ID3) DT algorithm uses information gain."

$$\mathrm{Info(D)} = -\sum_{i=1}^{m} \mathrm{pi} \log_2 \mathrm{pi} \tag{3.6}$$

$$\mathrm{Info(D)} = \sum_{i=1} \mathrm{Info(D)} - \mathrm{Info_A(D)} \tag{3.7}$$

Where, Pi is the probability that an arbitrary tuple in D belongs to class Ci.

$$\mathrm{Info_A(D)} = \sum_{j=1}^{v} \frac{|\mathrm{D}_j|}{|\mathrm{D}|} \times \mathrm{Info}(\mathrm{D}_j) \tag{3.8}$$

$$\mathrm{Info(A)} = \mathrm{Info(D)} - \mathrm{Info_A(D)} \tag{3.9}$$

$$\mathrm{Gain(A)} = \mathrm{Info(D)} - \mathrm{Info_A(D)} \tag{3.10}$$

Where,

- Info(D) is the average amount of information needed to identify the class label of a tuple in D.
- $\frac{|Dj|}{|D|}$ acts as the weight of the jth partition.
- InfoA(D) is the expected information required to classify a tuple from D based on the partitioning by A.

The attribute A with the highest information gain, Gain(A), is chosen as the splitting attribute at node N().

Information gain is based on the attribute with the most outcomes,the attribute with higher number of distinct values is preferred.For example , customer_ID as it have a unique value has a value of 0 in info(D) because of partition, in this way the information gain is maximized.[5]

$$\mathrm{SplitInfo_A(D)} = -\sum_{j=1}^{v} \frac{|\mathrm{D}_j|}{|\mathrm{D}|} \times \log_2\left(\frac{|\mathrm{D}_j|}{|\mathrm{D}|}\right) \tag{3.11}$$

Where,

- $\frac{|Dj|}{|D|}$ acts as the weight of the $j$th partition.
- v is the number of discrete values in attribute A.

The gain ratio can be defined as presented in Eq. 3.12.

$$\mathrm{GainRatio(A)} = \frac{\mathrm{Gain(A)}}{\mathrm{SplitInfo_A(D)}} \tag{3.12}$$

The attribute with the highest gain ratio is chosen as the splitting attribute.

Another DT algorithm CART (Classification and Regression Tree) uses the Gini method to create split points, as presented in Eq.3.13

$$\mathrm{Gini(D)} = 1 - \sum_{i=1}^{m} \mathrm{Pi}^2 \tag{3.13}$$

Where, Pi is the probability that a tuple in D belongs to class Ci.

As explained in [5], "The Gini Index considers a binary split for each attribute. You can compute a weighted sum of the impurity of each partition. If a binary split on attribute A partitions data D into D1 and D2, the Gini index of D is obtained by using Eq. 3.14."

$$\mathrm{Gini_A(D)} = \frac{|\mathrm{D1}|}{|\mathrm{D}|}\mathrm{Gini(D1)} + \frac{|\mathrm{D2}|}{|\mathrm{D}|}\mathrm{Gini(D2)} \tag{3.14}$$

As in [5],"In case of a discrete-valued attribute, the subset that gives the minimum gini index for that chosen is selected as a splitting attribute. In the case of continuous-valued attributes, the strategy is to select each pair of adjacent values as a possible split-point and point with smaller gini index chosen as the splitting point, as presented in Eq. 3.15."

$$\Delta\mathrm{Gini(A)} = \mathrm{Gini(D)} - \mathrm{Gini_A(D)} \tag{3.15}$$

The attribute with minimum Gini index is chosen as the splitting attribute.

## 3.3 Datasets Information

One of the key components of an investigation such as this is ... data. The dataset used has to be the most complete, up-to-date and thorough available. It should reflect the actual network flows of today, as well as identify the most possible amount of attack types, the most current ones and legacy ones. This was one of the defining steps of the investigation, and taking that on account we came across and gave a brief analysis to the following datasets:

- DARPA (Lincoln Laboratory - 1998, 1999);
- KDD'99 (University of California, Irvine - 1998, 99);
- DEFCON (The Shmoo Group - 2000);
- CAIDA (Center of Applied Internet Data Analysis – 2002/2016);
- LBNL (Lawrence Berkeley National Laboratory and ICSI – 2004/2005);
- CICIDS2017 (University of New Brunswick - Canadian Institute for Cybersecurity - 2017)
- CDX (United States Military Academy - 2009);
- Kyoto (Kyoto University – 2009);
- Twente (University of Twente – 2009);

- UMASS (University of Massachusetts – 2011);
- ISCX2012 (University of New Brunswick – 2012);
- ADFA (University of New South Wales – 2013);
- CIDDS (Coburg University of Applied Sciences- 2017)

The assessment of the (thirteen) most used relevant datasets since 1998 showed that most of them are outdated and tend to be unreliable. Some of these datasets suffer from a lack of diversity and traffic volumes, some do not cover the variety of known attacks, and others also lack features and metadata. The most promising datasets where the following:

- CIDDS-001[17] - is a labelled flow-based data set for evaluation of anomaly-based NIDS. For creation of the CIDDS-001 data set, a small business environment was emulated using OpenStack. This environment includes several clients and typical servers like an E-Mail server or a Web server. Python scripts emulate normal user behaviour on the clients. The CIDDS-001 contains unidirectional NetFlow data. It identifies the following attacks:
  - Dos
  - Brute force
  - PortScan
  - PingScan
- KD99 [8] - this is the data set used for The Third International Knowledge Discovery and Data Mining Tools Competition, which was held in conjunction with KDD-99 The Fifth International Conference on Knowledge Discovery and Data Mining. The competition task was to build a network intrusion detector, a predictive model capable of distinguishing between "bad" connections, called intrusions or attacks, and "good" normal connections. This database contains a standard set of data to be audited, which includes a wide variety of intrusions simulated in a military network environment. But the downside is that this dataset is quite old and does not overlook some of the modern attacks.
- CICIDS2017 - The Canada Institute for Cybersecurity (CIC) has made available a state-of-the-art dataset called CICIDS2017 [6], composed of the latest threats and

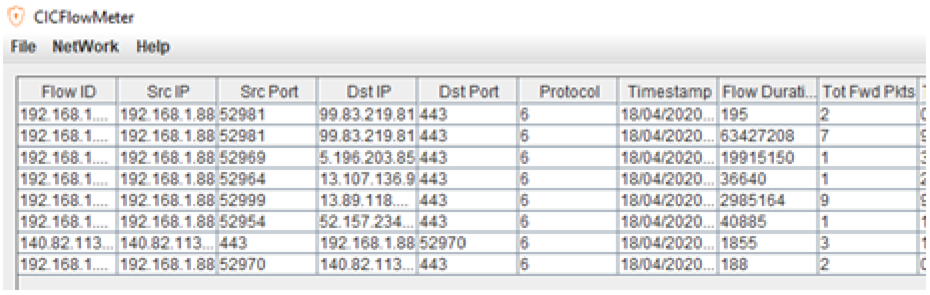
resources. The dataset draws the attention of many researchers, as it represents threats that were not addressed by the older datasets. It is currently considered to be the most complete available, and it has been widely used in research projects and even in the implementation of security solutions. The CICIDS2017 dataset contains common up-to-date attacks:

- DoS - GoldenEye, Slowloris, SlowHTTPTest and Hulk
- DDoS
- PortScan
- FTP - Patator
- SSH - Patator
- Bot
- Web Attacks - Brute Force, XSS & Sql Injection
- Infiltration
- Heartbleed

These attacks resemble the real data in the real world (in PCAP format). It presents the results of the network traffic analysis using the CICFlowMeter[14] tool, with flows labeled based on the date/time stamp, source and destination IPs, source and destination ports, protocols and attacks, in a Comma-separated Values (CSV) format file. The dataset was developed by Iman Sharafaldin, Arash Habibi Lashkari and Ali A. Ghorbani, and presented at the panel "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization", at the 4th International Conference on Information Systems Security and Privacy (ICISSP), Portugal, January 2018.

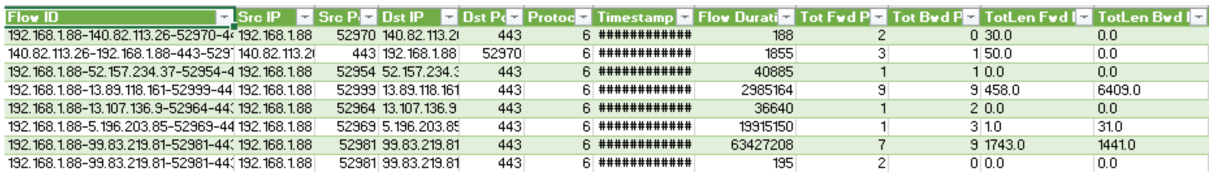
Without a doubt that the chosen dataset had to be the CICIDS2017, being one of the most (if not the most) complete and recent network intrusion datasets, and definitely the closest to perfect real-life data representation and classification. Its only downside is that since the attacks where grouped by type in specific days and hours windows, this dataset can not ever be used to prediction of new attacks, but to detection of attacks. It is our opinion that, in order to get predictions of new attacks, on a time basis based on time series, one should have a dataset with non-source-planned attacks and that is thoroughly studied and classified. This means to acomplish something that, for now, it stands as a almost impossible (extremely resource expensive) task.

The CICIDS2017 dataset is no more than results obtained/captured through a tool called CICFlowMeter[14], but properly attack-classified. This tool is a network traffic flow data generating software, made available by CIC, to generate 84 network traffic resources. It reads network physical adapters as well as PCAP files and generates a tabled report of the extracted resources while also providing report files in a CSV format. It is an open source software written in Java, it can be downloaded from Github and it was developed by the exact same team of the dataset itself. CICFlowMeter generates bidirectional flows (biflow), where the first packet determines the forward / backward (origin to destination) and backward / backward (destination to origin) directions, therefore, the 84 statistical resources, such as Duration, Number of packages, Number bytes, Packet length, etc are also calculated separately in the forward / backward and backward / backward directions. Information output is in CSV file format with six columns labeled for each stream, FlowID, SourceIP, destinationIP, SourcePort, DestinationPort and Protocol, and about 80 features of network traffic. It was observed that the TCP flows are usually terminated after the break of the connection (with sending a packet with FIN flag active), while the UDP flows are terminated by a flow timeout. The flow timeout value can be assigned arbitrarily by the individual scheme, for example, 600 seconds for TCP and UDP. This tool was properly installed, configured and explored, with results as seen in figure 3.9 ,considered positive and relevant to the development of the project. The figure 3.8 below shows the results captured after a simple visit to two public web pages.



| Flow ID | Src IP | Src Port | Dst IP | Dst Port | Protocol | Timestamp | Flow Durati... | Tot Fwd Pkts |
|---|---|---|---|---|---|---|---|---|
| 192.168.1.... | 192.168.1.88 | 52981 | 99.83.219.81 | 443 | 6 | 18/04/2020... | 195 | 2 |
| 192.168.1.... | 192.168.1.88 | 52981 | 99.83.219.81 | 443 | 6 | 18/04/2020... | 63427208 | 7 |
| 192.168.1.... | 192.168.1.88 | 52969 | 5.196.203.85 | 443 | 6 | 18/04/2020... | 19915150 | 1 |
| 192.168.1.... | 192.168.1.88 | 52964 | 13.107.136.9 | 443 | 6 | 18/04/2020... | 36640 | 1 |
| 192.168.1.... | 192.168.1.88 | 52999 | 13.89.118.... | 443 | 6 | 18/04/2020... | 2985164 | 9 |
| 192.168.1.... | 192.168.1.88 | 52954 | 52.157.234... | 443 | 6 | 18/04/2020... | 40885 | 1 |
| 140.82.113... | 140.82.113... | 443 | 192.168.1.88 | 52970 | 6 | 18/04/2020... | 1855 | 3 |
| 192.168.1.... | 192.168.1.88 | 52970 | 140.82.113... | 443 | 6 | 18/04/2020... | 188 | 2 |

Figure 3.8: CicFlowMeter capturing network flows



| Flow ID | Src IP | Src P | Dst IP | Dst P | Protoc | Timestamp | Flow Durati | Tot Fwd P | Tot Bwd P | TotLen Fwd | TotLen Bwd |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 192.168.1.88-140.82.113.26-52970-4 | 192.168.1.88 | 52970 | 140.82.113.2 | 443 | 6 | ############ | 188 | 2 | 0 | 30.0 | 0.0 |
| 140.82.113.26-192.168.1.88-443-529 | 140.82.113.2 | 443 | 192.168.1.88 | 52970 | 6 | ############ | 1855 | 3 | 1 | 50.0 | 0.0 |
| 192.168.1.88-52.157.234.37-52954-4 | 192.168.1.88 | 52954 | 52.157.234.3 | 443 | 6 | ############ | 40885 | 1 | 1 | 0.0 | 0.0 |
| 192.168.1.88-13.89.118.161-52999-44 | 192.168.1.88 | 52999 | 13.89.118.161 | 443 | 6 | ############ | 2985164 | 9 | 9 | 458.0 | 6409.0 |
| 192.168.1.88-13.107.136.9-52964-44 | 192.168.1.88 | 52964 | 13.107.136.9 | 443 | 6 | ############ | 36640 | 1 | 2 | 0.0 | 0.0 |
| 192.168.1.88-5.196.203.85-52969-44 | 192.168.1.88 | 52969 | 5.196.203.8 | 443 | 6 | ############ | 19915150 | 1 | 3 | 1.0 | 31.0 |
| 192.168.1.88-99.83.219.81-52981-44 | 192.168.1.88 | 52981 | 99.83.219.81 | 443 | 6 | ############ | 63427208 | 7 | 9 | 1743.0 | 1441.0 |
| 192.168.1.88-99.83.219.81-52981-44 | 192.168.1.88 | 52981 | 99.83.219.81 | 443 | 6 | ############ | 195 | 2 | 0 | 0.0 | 0.0 |

Figure 3.9: Excel sheet with imported CSV file data captured by CicFlowMeter

# Chapter 4

# Results and Analysis

After developing the structure for model training, a set of tests was defined to obtain results that allow us to conclude which algorithm is more appropriate, the one(s) with better performance, times (training and evaluation) and size, for each type of attack and for the combined set of attacks. After obtaining the results for these questions and knowing the best algorithms, the model for all the attacks can be used in a generic/general classification, and in case of doubt or uncertainty, an ideally deeper and more accurate analysis and classification can be made with the most adequate algorithm for a certain type of attack. Models where trained for each type of attack/dataset, and they were evaluated, tested and this tests were also evaluated. The tests were made with changes of the following parameters:

- Random state - defined when the dataset is splitted into a training set and a test set, this parameter is responsible for data order randomization, promoting a more impartial use and less subjective to standardization;
- Cross validation folds - this is a procedure that avoids two problems: overfitting and underfitting. Overfitting happens when to much data noise is processed due to excessive amount of data. Underfitting takes place when the data is not enough or was not well interpreted/fitted. Cross validation gives us an estimate of the accuracy of the model by parceling a part of the training dataset as test set and uses it to self evaluate the rest. This parcel is turned over until all the data was used as test set. One full cycle of these iterations is a fold. The more folds, the more resources and time is needed for the processing.

A dataset containing all the type of previous mentioned attacks was created and named as "ALL". The ALL dataset has not been subjected to different tests because its size makes tests take an abnormal amount of time when the parameter values are significantly higher than the single test made. The test parameters for this dataset were 10 cross validation folds and state random also of 10. For the datasets of each type of attack, 4 different tests were then defined and conducted:

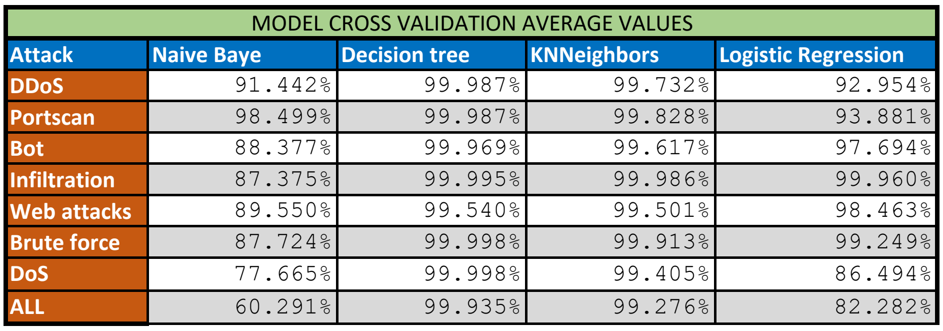
- A - 10 Cross validation folds & No state random
- B - 50 Cross validation folds & 10 State random
- C - 100 Cross validation folds & 20 State random
- D - 500 Cross validation folds & 20 State random

Each of these tests was performed on each of the datasets and for each of the 4 algorithms, except for the ALL dataset that did not went through test D due to the lack of hardware resources, having all registered values grouped by algorithm and divided into two tables: one with the results of the cross validation score and the other with the results of the accuracy obtained with the test data. From these tables, averages and standard deviations were then calculated, and the maximums were identified. Then, two tables (4.1 and 4.1) were created with the average values previously calculated.

## 4.1 Performance

Performance tests where conducted according to four different evaluation methods: cross validation, accuracy, confusion matrix and classification report. But for the objective of this investigation the relevant evaluation method needed is the accuracy, which can be obtained through the its own method during the test data test phase, but when it comes to model accuracy estimation its recommended to use cross validation (for the reasons explaind on the chapter introduction). The following table 4.1 represents the cross validation average scores of the tests A, B, C and D, for each attack and each algorithm. It is observed that NB algorithm has the lowest values, followed by Logistic Regression. DT and KNN seem to have similar scores with all of them over 99%, which represents excellent expected accuracy of the models.

Fig 4.2 shows tables 4.1 data. It helps us visualize the results and right away pops out that NB has the lowest scores and that DT and KNN seem head-to-head.



| MODEL CROSS VALIDATION AVERAGE VALUES | | | | |
|---|---|---|---|---|
| Attack | Naive Baye | Decision tree | KNNeighbors | Logistic Regression |
| DDoS | 91.442% | 99.987% | 99.732% | 92.954% |
| Portscan | 98.499% | 99.987% | 99.828% | 93.881% |
| Bot | 88.377% | 99.969% | 99.617% | 97.694% |
| Infiltration | 87.375% | 99.995% | 99.986% | 99.960% |
| Web attacks | 89.550% | 99.540% | 99.501% | 98.463% |
| Brute force | 87.724% | 99.998% | 99.913% | 99.249% |
| DoS | 77.665% | 99.998% | 99.405% | 86.494% |
| ALL | 60.291% | 99.935% | 99.276% | 82.282% |

Figure 4.1: Model cross validation average values

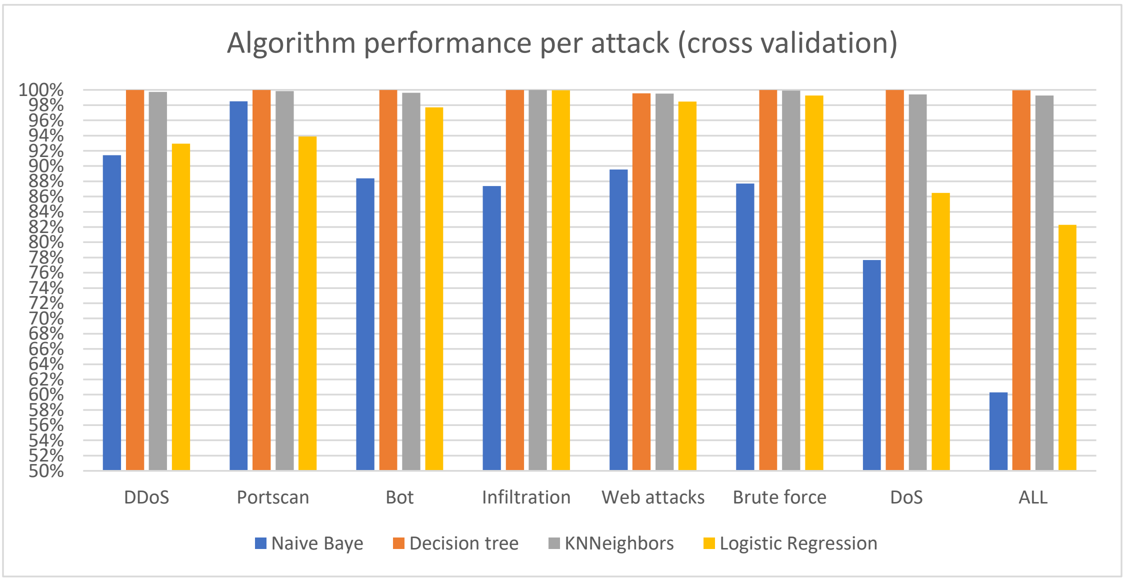


Figure 4.2: Algorithm performance per attack (cross validation)

Fig 4.3 represents the top 3% of the previous fig 4.2, as if this was zoomed. NB barely makes it to the fig and we can see that DT actually out-performs KNN in every type of attack.

On table 4.1 we have the results of the tests made with the test dataset, for accuracy. With no surprise, all the values seem very close to those obtained through cross validation. NB, again, has the lowest scores, and DT and KNN seem very close to each other.

With the observation of fig 4.4, which was generated from the previous table data, we confirm what was seen before, with NBs with values far from the others, and the competition repeats itself with DT and KNN, despite that on this fig we can immediately see that DT has equal or better scores than its rival KNN.

In fact, as shown on table 4.5, DT scores pass the 99,9% mark except for Web Attacks,

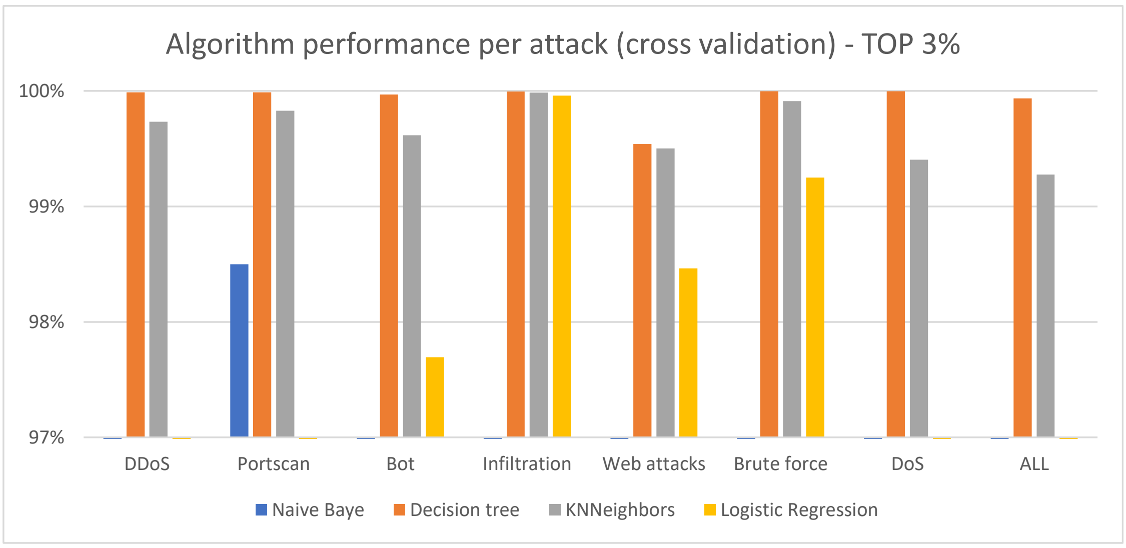


Figure 4.3: Algorithm performance per attack (cross validation) - TOP 3%

| TEST ACCURACY AVERAGE VALUES | | | | |
|---|---|---|---|---|
| **Attack** | **Naive Baye** | **Decision tree** | **KNNeighbors** | **Logistic Regression** |
| **DDoS** | 91.421% | 99.987% | 99.732% | 93.066% |
| **Portscan** | 99.220% | 99.988% | 99.834% | 95.326% |
| **Bot** | 88.373% | 99.973% | 99.588% | 97.663% |
| **Infiltration** | 87.327% | 99.998% | 99.988% | 99.968% |
| **Web attacks** | 89.574% | 99.513% | 99.483% | 98.323% |
| **Brute force** | 87.724% | 99.998% | 99.930% | 99.359% |
| **DoS** | 77.669% | 99.936% | 99.416% | 86.491% |
| **ALL** | 60.120% | 99.809% | 99.299% | 81.835% |

Table 4.1: Test accuracy average values

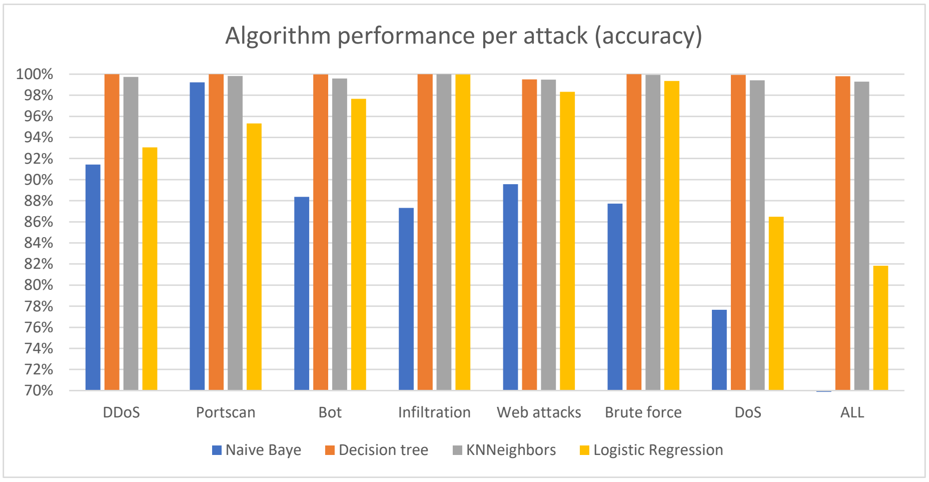
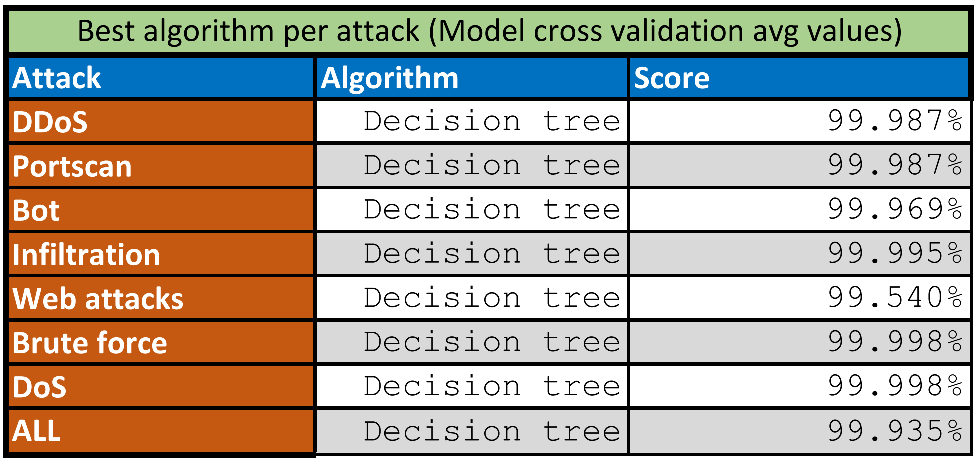


Figure 4.4: Algorithm performance per attack (accuracy)

getting a slight lower score of 99,5%, confirming that according to cross validation, this is the best algorithm for all the scenarios. Fig. 4.6 demonstrates how close the scores are to 100%.



| Best algorithm per attack (Model cross validation avg values) | | |
|---|---|---|
| **Attack** | **Algorithm** | **Score** |
| **DDoS** | Decision tree | 99.987% |
| **Portscan** | Decision tree | 99.987% |
| **Bot** | Decision tree | 99.969% |
| **Infiltration** | Decision tree | 99.995% |
| **Web attacks** | Decision tree | 99.540% |
| **Brute force** | Decision tree | 99.998% |
| **DoS** | Decision tree | 99.998% |
| **ALL** | Decision tree | 99.935% |

Figure 4.5: Best algorithm per attack (model cross validation average values)

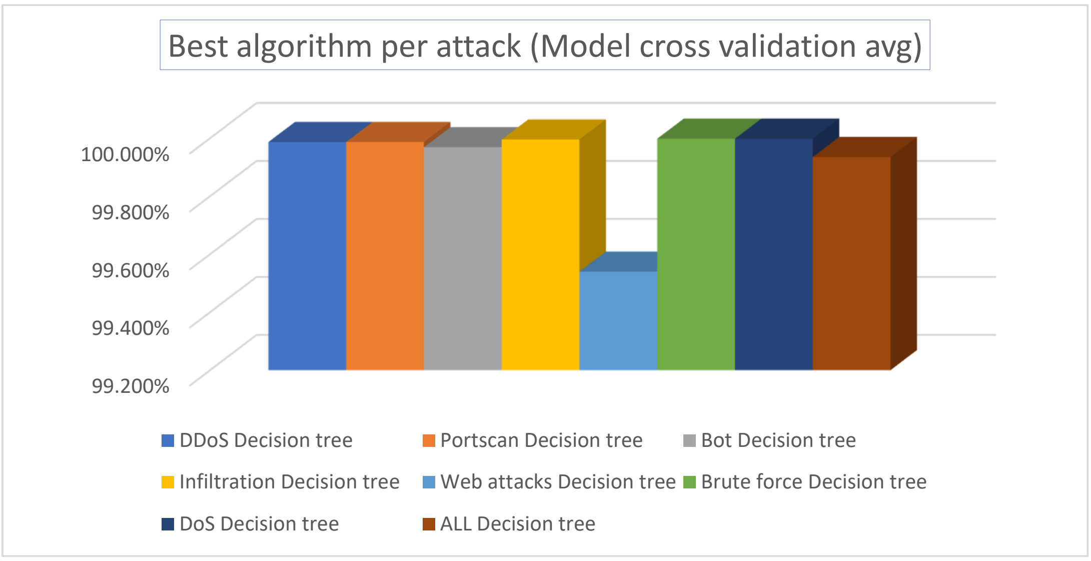
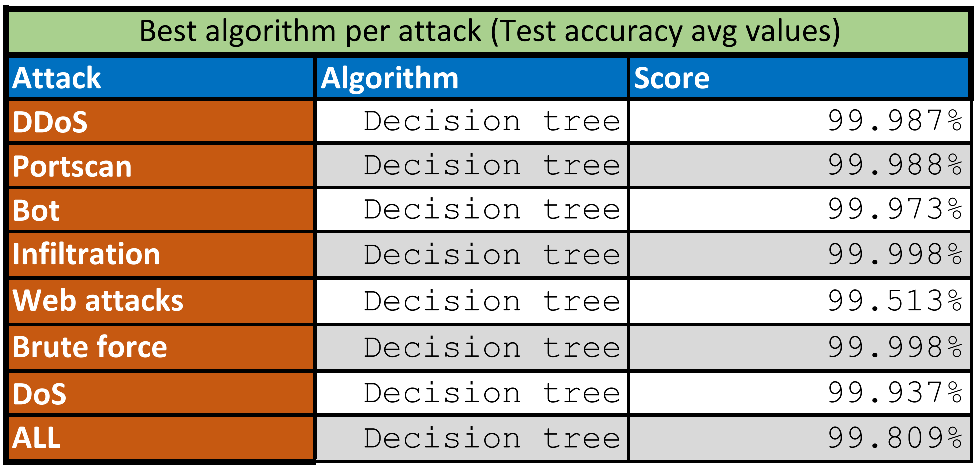


Figure 4.6: Best algorithm per attack (model cross validation avg)

When it comes to the accuracy, the score for test data classification represented on table 4.2 we have DT as the most accurate algorithm again on all types of attacks and for the ALL attack dataset. Fig 4.7 reveals that there was a slight cutback in the ALL dataset, but still its score reaches 99,8% and so its accuracy is very good.



| Best algorithm per attack (Test accuracy avg values) | | |
|---|---|---|
| Attack | Algorithm | Score |
| DDoS | Decision tree | 99.987% |
| Portscan | Decision tree | 99.988% |
| Bot | Decision tree | 99.973% |
| Infiltration | Decision tree | 99.998% |
| Web attacks | Decision tree | 99.513% |
| Brute force | Decision tree | 99.998% |
| DoS | Decision tree | 99.937% |
| ALL | Decision tree | 99.809% |

Table 4.2: Best algorithm (test accuracy avg values) - TABLE

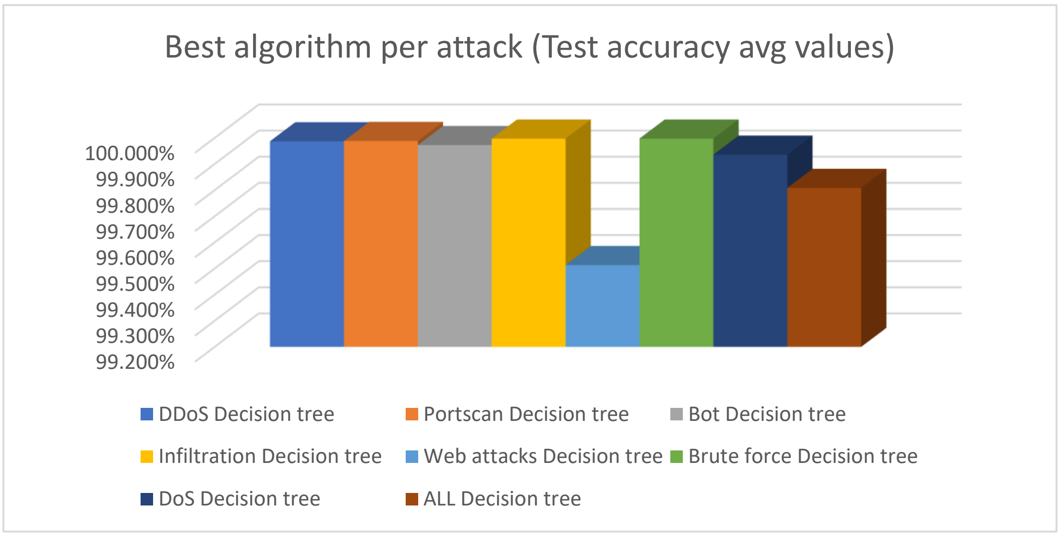
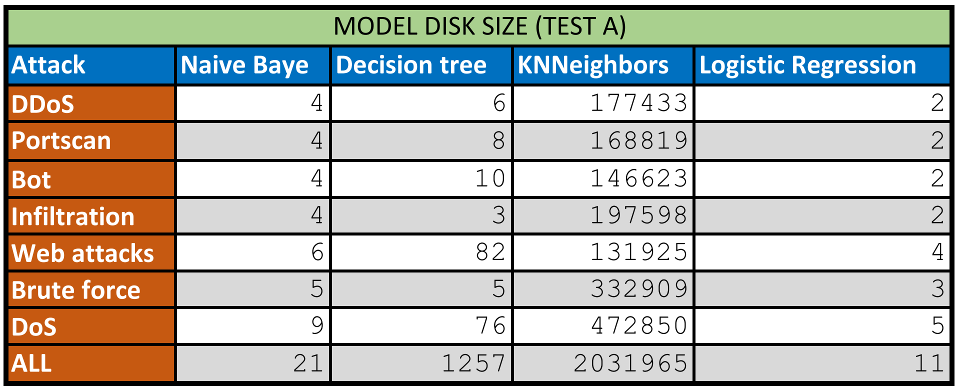


Figure 4.7: Best algorithm per attack (test accuracy avg values) - CHART

## 4.2 Generated Models Size

In this section there will be analyzed the machine learning model size in disk that each algorithm outputs.

Regarding the physical size of the models, registered in table 4.8, we can see that Logistic Regression is the algorithm that generates the lowest size model. On the other hand, K-N-Neighbors is the algorithm that generates, by far, the largest model with more than 2GB and its smallest model reaches 130MB. This means the its size scale is completely different than the rest of the algorithms. In some situations KNN seems to be the correct alternative do DT, but since its model is exponentially bigger that the latter, the classification of new data would take much longer to be obtained and take up quite some more resources. Size of model has strong impact on the algorithm decision due to bigger models taking longer to make decisions. In the fig 4.9 we can have a more clear idea of the differences in models size. This fig was limited at 200KB and KNN was not included since its models sizes invalidate it's use. Again, despite the clear difference in Web Attacks, DoS and the ALL dataset, DT continues to be the correct choice for most scenarios, with suitable sized models (no more than 100KB) for separate attack types, and a 1.2MB model for the ALL dataset.



| MODEL DISK SIZE (TEST A) | | | | |
|---|---|---|---|---|
| Attack | Naive Baye | Decision tree | KNNeighbors | Logistic Regression |
| DDoS | 4 | 6 | 177433 | 2 |
| Portscan | 4 | 8 | 168819 | 2 |
| Bot | 4 | 10 | 146623 | 2 |
| Infiltration | 4 | 3 | 197598 | 2 |
| Web attacks | 6 | 82 | 131925 | 4 |
| Brute force | 5 | 5 | 332909 | 3 |
| DoS | 9 | 76 | 472850 | 5 |
| ALL | 21 | 1257 | 2031965 | 11 |

Figure 4.8: Model physical disk size in KB

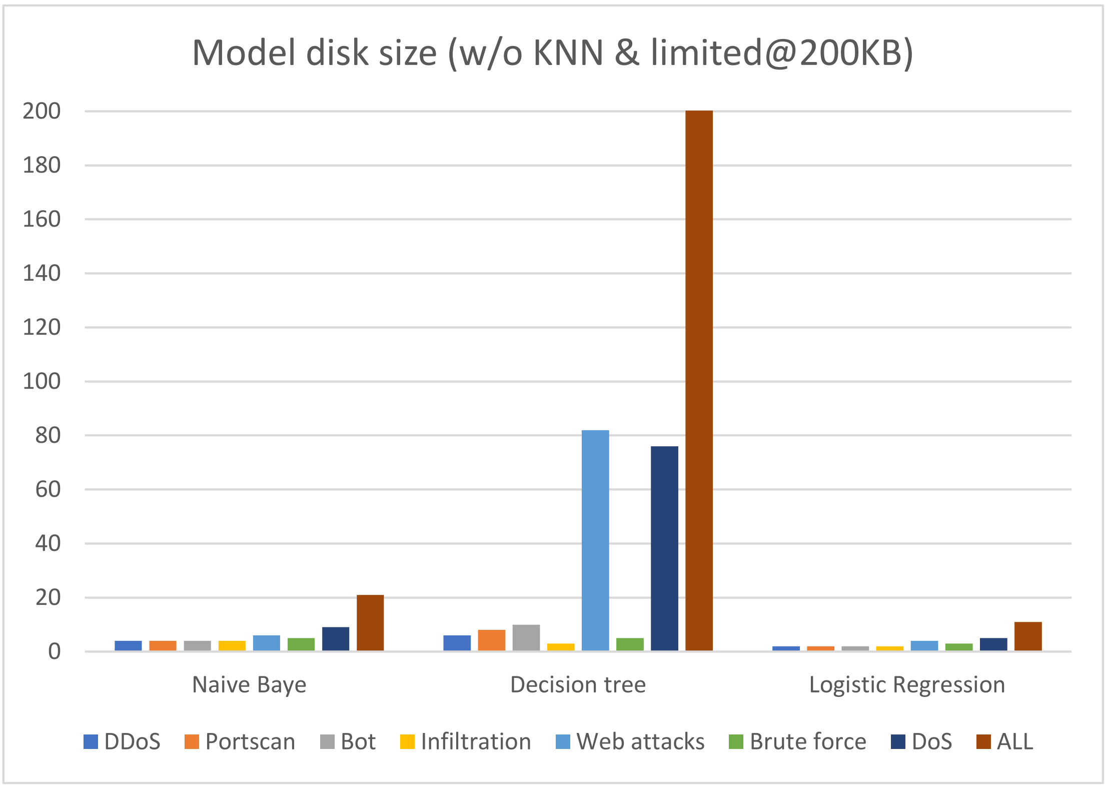
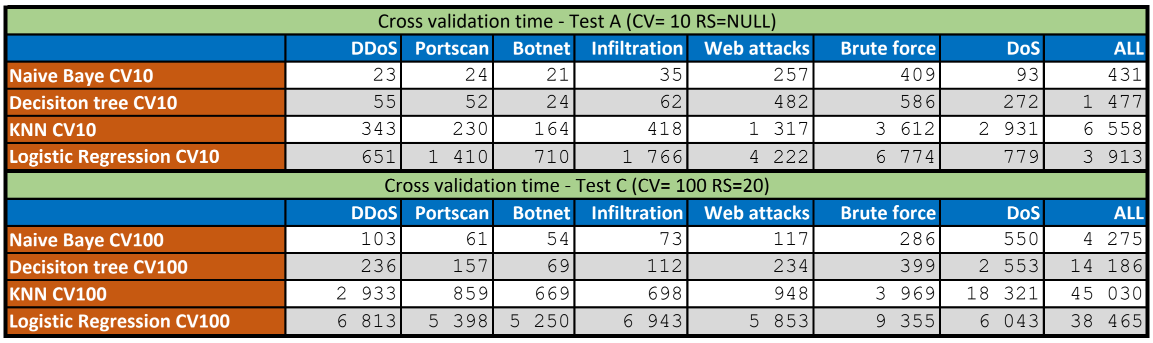


Figure 4.9: Model physical disk size in KB

## 4.3 Elapsed Time

Model training is usually not conditioned by the time the models take to be completed. Models tend to be trained in advance, and in fact what is intended is high performance models with small sizes, so that predictions/classifications are fast and accurate. However, it was considered pertinent for this report, considering the scope of this investigation, the recording and analysis of some of the times taken to train and test the models. Thus, cross validation times and model fit times were recorded for tests A and C, for each of the datasets and algorithms. In table 4.10 and fig 4.11 we can observe cross validation times. Despite that KNN has the biggest sized models, its cross validation times are significantly lower than the ones from Logistic Regression, but nonetheless it still took quite longer than NB and DT. So, once again, DT seems to be the right choice (since performance of NB is poor). The ALL dataset has the longest times, which comes without surprises, reaching (more than) 12,5 hours with KNN.



| Cross validation time - Test A (CV= 10 RS=NULL) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | DDoS | Portscan | Botnet | Infiltration | Web attacks | Brute force | DoS | ALL |
| Naive Baye CV10 | 23 | 24 | 21 | 35 | 257 | 409 | 93 | 431 |
| Decisiton tree CV10 | 55 | 52 | 24 | 62 | 482 | 586 | 272 | 1 477 |
| KNN CV10 | 343 | 230 | 164 | 418 | 1 317 | 3 612 | 2 931 | 6 558 |
| Logistic Regression CV10 | 651 | 1 410 | 710 | 1 766 | 4 222 | 6 774 | 779 | 3 913 |
| Cross validation time - Test C (CV= 100 RS=20) | | | | | | | | |
| | DDoS | Portscan | Botnet | Infiltration | Web attacks | Brute force | DoS | ALL |
| Naive Baye CV100 | 103 | 61 | 54 | 73 | 117 | 286 | 550 | 4 275 |
| Decisiton tree CV100 | 236 | 157 | 69 | 112 | 234 | 399 | 2 553 | 14 186 |
| KNN CV100 | 2 933 | 859 | 669 | 698 | 948 | 3 969 | 18 321 | 45 030 |
| Logistic Regression CV100 | 6 813 | 5 398 | 5 250 | 6 943 | 5 853 | 9 355 | 6 043 | 38 465 |

Figure 4.10: Cross validation times in seconds

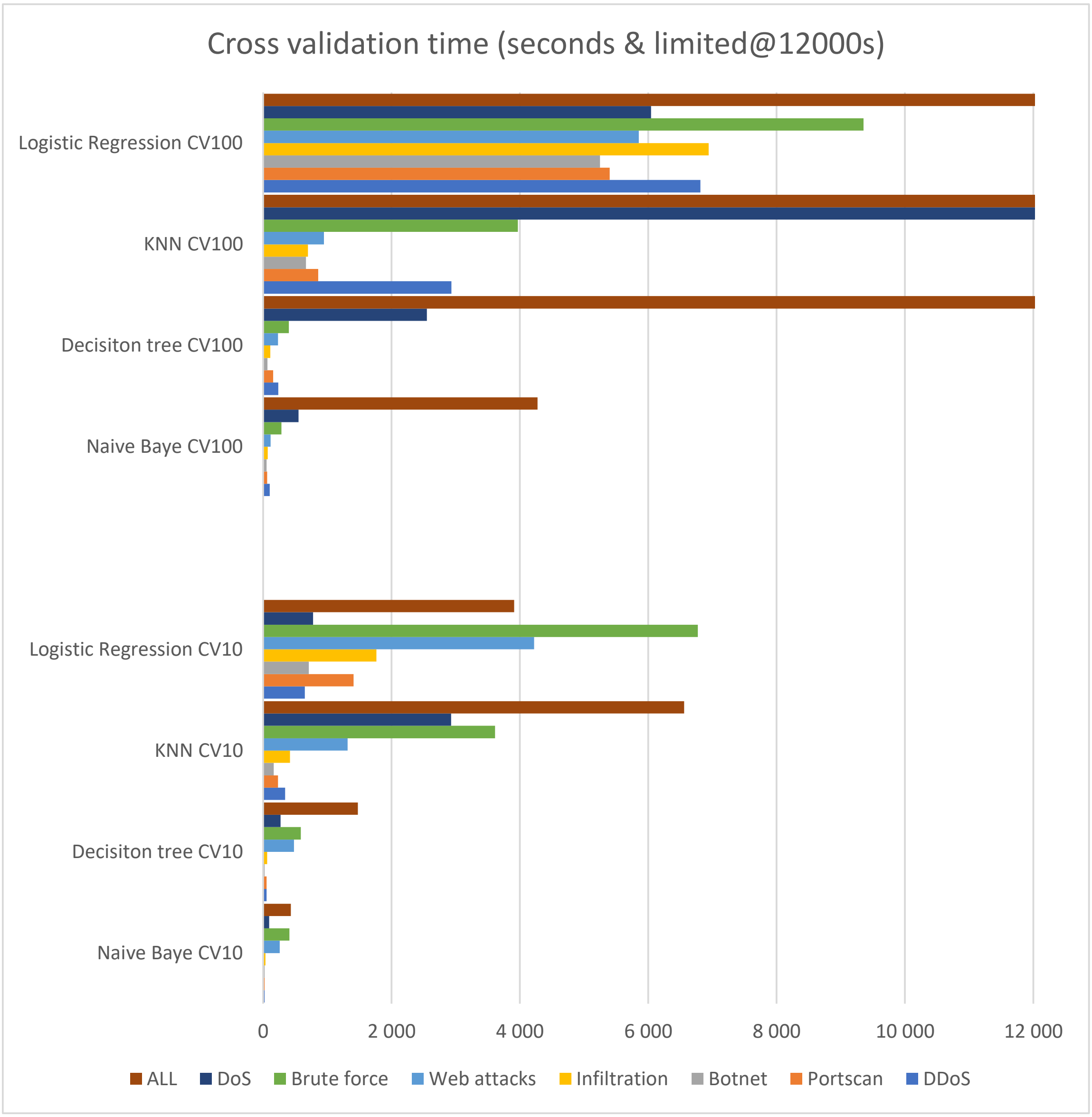
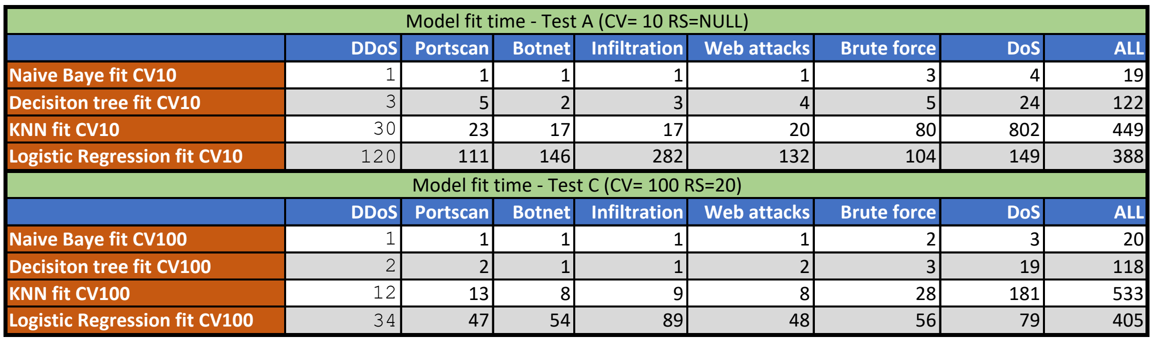


Figure 4.11: Cross validation times in seconds

As far as the fitting times concern, and accordingly to table 4.12 and fig 4.13, considering that the performance of DT is excellent, a normal fit revealed that this procedure is quite fast and without the need for further fitting iterations. KNN and LR are the most fitting time expensive algorithms, while NB and DT are the least expensive. So, and keeping the pattern, DT is still at the nose of the competition. It is noticeable that the ALL dataset doesn't extend fit times on an exponential level like it does in cross validation, having acceptable times for any of the algorithms. The longest time registered was (as always) with KNN with 533 seconds (approximately 9 minutes).



| Model fit time - Test A (CV= 10 RS=NULL) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | DDoS | Portscan | Botnet | Infiltration | Web attacks | Brute force | DoS | ALL |
| Naive Baye fit CV10 | 1 | 1 | 1 | 1 | 1 | 3 | 4 | 19 |
| Decisiton tree fit CV10 | 3 | 5 | 2 | 3 | 4 | 5 | 24 | 122 |
| KNN fit CV10 | 30 | 23 | 17 | 17 | 20 | 80 | 802 | 449 |
| Logistic Regression fit CV10 | 120 | 111 | 146 | 282 | 132 | 104 | 149 | 388 |
| Model fit time - Test C (CV= 100 RS=20) | | | | | | | | |
| | DDoS | Portscan | Botnet | Infiltration | Web attacks | Brute force | DoS | ALL |
| Naive Baye fit CV100 | 1 | 1 | 1 | 1 | 1 | 2 | 3 | 20 |
| Decisiton tree fit CV100 | 2 | 2 | 1 | 1 | 2 | 3 | 19 | 118 |
| KNN fit CV100 | 12 | 13 | 8 | 9 | 8 | 28 | 181 | 533 |
| Logistic Regression fit CV100 | 34 | 47 | 54 | 89 | 48 | 56 | 79 | 405 |

Figure 4.12: Model fitting times in seconds

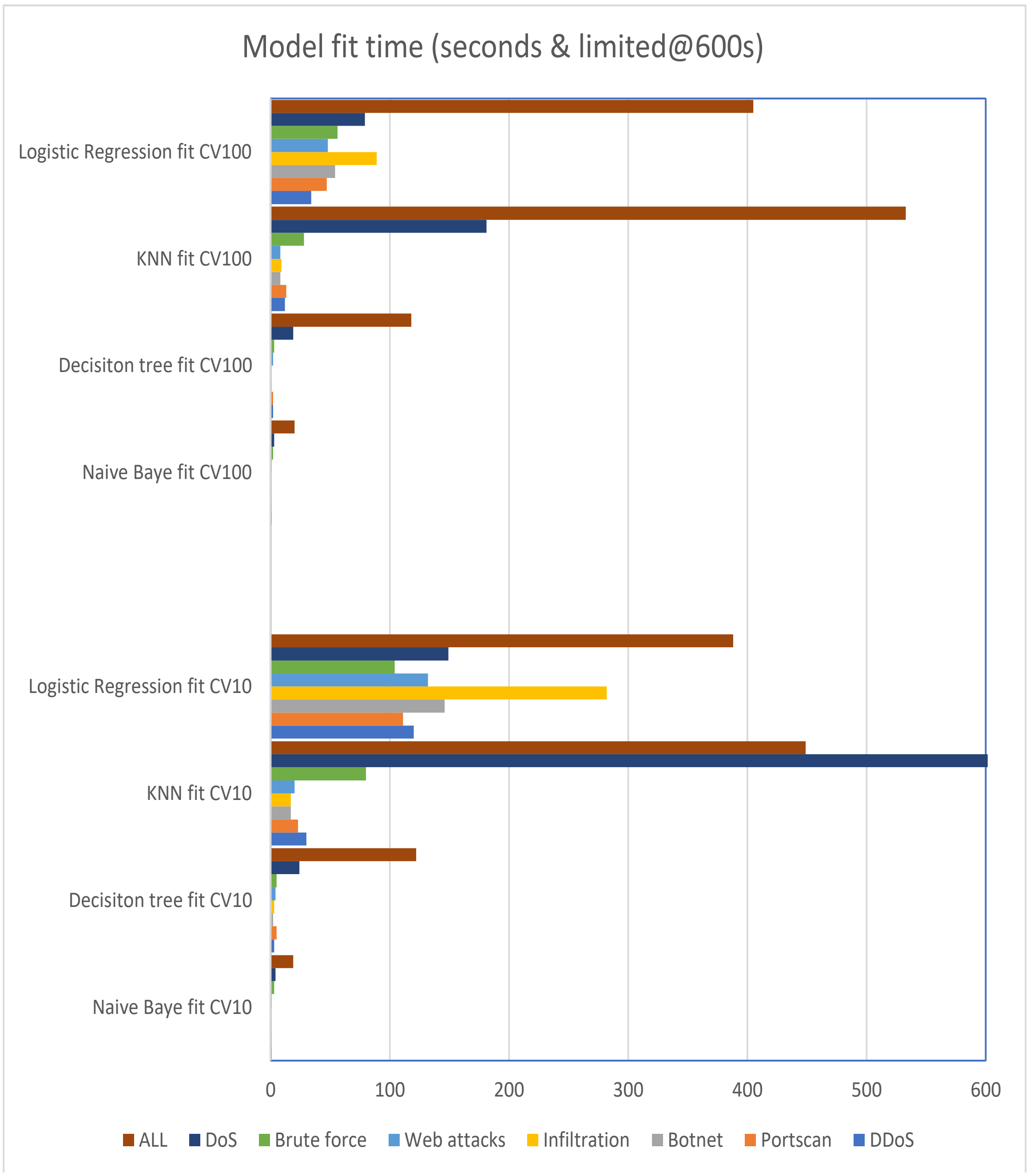


Figure 4.13: Model fitting times in seconds

Chapter 5

# Conclusion

In conclusion, the project was conducted with success, having reached the main goal, we were able to find the best performing algorithm, within the tested ones, to detect attacks present in the dataset.

Along the way this project made us improve and dig a bit more knowledge about machine learning as well as making us more aware of the work being done in the field of cybersecurity, which has the huge responsibility of protecting information the best possible way, in a never-slowing-down rhythm.

Based on our results and analysis the DT is definitely the best all around algorithm to use in this case. It obtained excellent performance in all types of attacks with accuracy more than 99%, combined with a decent sized model and with fast training and testing times, making it one of the best algorithms to do this type of classification.

This project proves that machine learning has huge potential for distinguishing malicious from non-malicious attempts and high percentage of attack detection, lowering the false positives that usual IDS tend to have, proving this way the benefits of using this technique.

# References

[1] *A Gentle Introduction to Bayes Theorem for Machine Learning - Machine Learning Mastery.* URL: https://machinelearningmastery.com/bayes-theorem-for-machine-learning/ (visited on 12/04/2019).

[2] *A Tour of Machine Learning Algorithms.* URL: https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/ (visited on 06/17/2020).

[3] Mohammed Hamid Abdulraheem and Najla Badie Ibraheem. "A DETAILED ANALYSIS OF NEW INTRUSION DETECTION DATASET". In: 2019.

[4] Mohammad Almseidin et al. "Evaluation of Machine Learning Algorithms for Intrusion Detection System". In: (Jan. 2018). arXiv: 1801.02330. URL: https://arxiv.org/abs/1801.02330.

[5] *Decision Tree Classification in Python - DataCamp.* URL: https://www.datacamp.com/community/tutorials/decision-tree-classification-python (visited on 06/24/2020).

[6] *IDS 2017 — Datasets — Research — Canadian Institute for Cybersecurity — UNB.* URL: https://www.unb.ca/cic/datasets/ids-2017.html (visited on 06/17/2020).

[7] Anirudh Janagam and Saddam Hossen. "Analysis of Network Intrusion Detection System with Machine Learning Algorithms (Deep Reinforcement Learning Algorithm)". In: 2018.

[8] *KDD Cup 1999 Data.* URL: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html (visited on 06/17/2020).

[9] Jiyeon Kim, Yulim Shin, and Eunjung Choi. "An Intrusion Detection Model based on a Convolutional Neural Network". In: *Journal of Multimedia Information System* 6 (2019), pp. 165–172. DOI: 10.33851/JMIS.2019.6.4.165.

[10] *KNN Classification using Scikit-learn - DataCamp.* URL: https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn (visited on 06/17/2020).

[11] *KNN Classification using Scikit-learn - DataCamp.* URL: https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn (visited on 06/24/2020).

[12] *Logistic Regression in R Tutorial - DataCamp.* URL: https://www.datacamp.com/community/tutorials/logistic-regression-R (visited on 06/24/2020).

[13] *Naive Bayes Classification using Scikit-learn - DataCamp.* URL: https://www.datacamp.com/community/tutorials/naive-bayes-scikit-learn (visited on 06/24/2020).

[14] *NetFlowMeter.* URL: http://netflowmeter.ca/ (visited on 06/17/2020).

[15] *Network Design: Firewall, IDS/IPS.* URL: https://resources.infosecinstitute.com/network-design-firewall-idsips/%7B%5C#%7Dgref (visited on 06/17/2020).

[16] Ranjit Panigrahi, Ranjit Panigrahi, and Samarjeet Borah. "A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems". In: *International Journal of Engineering & Technology* 7.3.24 (Aug. 2018), pp. 479–482. DOI: 10.14419/ijet.v7i3.24.22797. URL: https://www.sciencepubco.com/index.php/ijet/article/view/22797.

[17] Markus Ring et al. "Flow-based benchmark data sets for intrusion detection". In: 2017.

[18] Iman Sharafaldin, Arash Habibi Lashkari, and Ali Ghorbani. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization". In: 2018, pp. 108–116. DOI: 10.5220/0006639801080116.

[19] Ankit Thakkar and Ritika Lohiya. "A Review of the Advancement in Intrusion Detection Datasets". In: *Procedia Computer Science* 167 (2020), pp. 636–645. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2020.03.330. URL: http://www.sciencedirect.com/science/article/pii/S1877050920307961.

[20] *What is a honeypot? How honeypots help security — Kaspersky.* URL: https://usa.kaspersky.com/resource-center/threats/what-is-a-honeypot (visited on 06/17/2020).

[21] *What is an Intrusion Detection System? - Palo Alto Networks.* URL: https://www.paloaltonetworks.com/cyberpedia/what-is-an-intrusion-detection-system-ids (visited on 06/17/2020).

[22] *What Is an Intrusion Detection System? Definition, Types, and Tools - DNSstuff.* URL: https://www.dnsstuff.com/intrusion-detection-system (visited on 06/17/2020).

[23] Arif Yulianto, Parman Sukarno, and Novian Anggis Suwastika. "Improving AdaBoost-based Intrusion Detection System (IDS) Performance on CIC IDS 2017 Dataset". In: *Journal of Physics: Conference Series* 1192 (Mar. 2019), p. 012018. ISSN: 1742-6588. DOI: 10.1088/1742-6596/1192/1/012018. URL: https://iopscience.iop.org/article/10.1088/1742-6596/1192/1/012018.

[24] Qianru Zhou and Dimitrios Pezaros. "Evaluation of Machine Learning Classifiers for Zero-Day Intrusion Detection – An Analysis on CIC-AWS-2018 dataset". In: (May 2019). arXiv: 1905.03685. URL: http://arxiv.org/abs/1905.03685.