UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE
TECNOLOGÍAS Y SERVICIOS DE
TELECOMUNIACIÓN
TRABAJO FIN DE GRADO

DISEÑO DE UN MODELO EFICIENTE DE PREPROCESAMIENTO DE DATOS APLICADO A SISTEMAS DE DETECCIÓN DE INTRUSIONES BASADO EN TÉCNICAS DE DEEP LEARNING

DESIGN OF AN EFFICIENT DATA PREPROCESING MODEL APPLIED TO INTRUSION DETECTION SYSTEMS BASED ON DEEP LEARNING TECHNIQUES

BEATRIZ ESTEBAN NAVARRO
JUNIO 2019

GRADO EN TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIONES

TRABAJO FIN DE GRADO

Título:	Diseño de un modelo eficiente de preprocesamiento de datos aplicado a Sistemas de Detección de Intrusiones basado en técnicas de Deep Learning.
Title:	Design of an efficient data preprocessing model applied to Intrusion Detection Systems based on Deep Learning techniques.
Autor:	Beatriz Esteban Navarro
Tutor:	Xavier Larriva Novo
Ponente:	Víctor Abraham Villagrá González
Departamento:	Departamento de Ingeniería en Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL

Presidente: D. ……

Vocal: D. ……..

Secretario: D. ……..

Suplente: D. ……..

Los miembros del tribunal arriba nombrados acuerdan otorgar la calificación de: ……

Madrid, a de de 20…

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIONES
TRABAJO FIN DE GRADO

DISEÑO DE UN MODELO EFICIENTE DE PREPROCESAMIENTO DE DATOS APLICADO A SISTEMAS DE DETECCIÓN DE INTRUSIONES BASADO EN TÉCNICAS DE DEEP LEARNING

DESIGN OF AN EFFICIENT DATA PREPROCESING MODEL APPLIED TO INTRUSION DETECTION SYSTEMS BASED ON DEEP LEARNING TECHNIQUES

BEATRIZ ESTEBAN NAVARRO

JUNIO 2019

RESUMEN

Actualmente, los Sistemas de Detección de Intrusiones recogen una gran cantidad de datos relacionados con actividades peligrosas o no autorizadas en la red. Esta información puede ser utilizada por redes neuronales para crear modelos con el objetivo de predecir y clasificar posibles ataques y mejorar la tasa de falsos positivos.

El presente trabajo propone un modelo eficiente de preprocesamiento de datos aplicado a dichos sistemas, basándose en una Deep Learning Network.

Los datasets utilizados para la evaluación del modelo han sido el UGR16, el NSL-KDD y el UNSW-NB15.

Por una parte, se ha analizado el contenido de estos de tal manera que se ha diferenciado entre valores de tipo *no numérico* y tipo *numérico*. Así mismo, se han estudiado los diferentes métodos de preprocesamiento existentes, siendo la normalización *zscore* y *minmax* las utilizadas para este proyecto. Finalmente, se han aplicado de diversas maneras a cada dataset y se han obtenido múltiples resultados que han sido evaluados con las técnicas de *precisión de clasificación* y *matriz de confusión.*

Para llevar a cabo el proyecto, en primer lugar, se han transformado todos los valores a tipo numérico ya que de otra manera la red no puede funcionar. En segundo lugar, se han aplicado los métodos de preprocesamiento, mencionados en el párrafo anterior, a cada dataset por separado. Por último, se han evaluado los resultados con las técnicas de evaluación ya mencionadas.

ABSTRACT

Nowadays, Intrusion Detection Systems collect a large amount of data related to dangerous or unauthorized activities in networks. This information can be used by neural networks to create models with the objective of predicting attacks and improve the rate of false positives.

This paper proposes an efficient model of data preprocessing applied to these systems, based on a Deep Neural Network.

The datasets used for the evaluation of the model have been the UGR16, the NSL-KDD and the UNSW-NB15. On the one hand, the content of these has been analysed in such a way that a distinction has been made between values of a *non-numerical* type and a *numerical* type. Likewise, the different existing preprocessing methods have been studied, being *zscore* and *minmax* normalization those used in this project. Finally, they have been applied in different ways to each dataset and multiple results have been obtained that have been evaluated with the techniques of *classification accuracy* and *confusion matrix*.

To carry out the project, in the first place, all the values have been transformed to numerical type as otherwise the network cannot work. Second, the preprocessing methods mentioned in the previous paragraph have been applied to each dataset separately. Finally, the results have been reviewed with the evaluation techniques already mentioned.

PALABRAS CLAVE

IDS, Detección de intrusiones, Machine Learning, Deep Learning, Deep Neural Network, Preprocesamiento de datos, UGR16, NSL-KDD, UNSW-NB15

KEYWORDS

IDS, Intrusion detection, Machine Learning, Deep Learning, Deep Neural Network, Data preprocessing, UGR16, NSL-KDD, UNSW-NB15

AGRADECIMIENTOS

A mi tutor, ponente y todas las personas que me han apoyado.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

IDS Intrusion Detection System

ML Machine Learning

AI Artificial intelligence

FS Feature selection

CS Case selection

KNN K-Nearest Neighbors

FCM Fuzzy C-means clustering

FE Fuzzy entropy clustering

PCA Principal component Analysis

SVM Support Vector Machine

ANN Artificial Neural Network

PSO WENN Particle swarm optimization weight extraction algorithm for a neural network classifier

BPNN Back Propagation Neural Network

BC Bayesian classifier

AC Associative classifier

DT Decision tree

CART Classification and regression trees

CBR Case-based reasoning

SIDS Stack based intrusion detection system

NIDS Network based intrusion detection system

HIDS Host based intrusion detection system

1. INTRODUCTION AND PROJECT GOALS

1.1. INTRODUCTION

Intrusion detection is an important task in cybersecurity, and thanks to Intrusion Detection Systems network activity can be monitored for malicious, unauthorized activity or dangerous threats.

Machine Learning and Deep Learning techniques makes possible to use the large quantities of data collected by IDS to develop models to predict attacks. The main characteristic of Intrusion Detection Systems based on machine learning is to make the algorithm learn and make predictions from an input-output pair. Among the types of machine learning, supervised learning is found offering *classification* model- categorical output, and *regression* model- numerical output. Among the classifiers, Deep Neural Networks are the most popular option for intrusion detection because of their tolerance to inaccurate data and uncertain information, and their ability to infer solutions.

When training these neural networks, data has to be prepared before using it, being an option to preprocess it. Preprocessing data consist on transforming the input values with diverse techniques to improve the performance of the network, having on the one hand, the conversion of non-numerical features to numerical, and on the other, the transformations applied to numerical data.

Non-numerical values can be transformed with *One-Hot encoding* methods: encoding to dummy variables, employed when the number of features to encode is high, or encoding to indexes, used when there is a reduced number of features to transform. For numerical data, normalization, that involves reducing the complexity of the network, accelerating the learning phase and making the network more efficient, is the main preprocessing technique and it is divided into *zscore* normalization and *min-max* normalization. *Zscore* or *standardization* consist of scaling the features to the standard normal distribution, whereas *min-max* normalization transforms the original range of values into a standard one, normally between 0 and 1 or -1 and 1.

There are many techniques for the deep neural network evaluation among which we find Classification Accuracy, Confusion Matrix, Precision and Recall, and Area Under the ROC Curve, using all of them the true/false positive and true/false negative rates given by the model.

This study is going to be developed with a classification deep neural network to evaluate which previously mentioned preprocessing technique is more accurate, evaluating the model with the Classification Accuracy and the Confusion Matrix metrics methods.

1.2. PROJECT GOALS

The aim of this project is to propose an efficient model of data preprocessing in neural networks using the information collected by Intrusion Detection Systems with the goal of decreasing the false positive rate and improving the accuracy.

To achieve this, it is necessary to:

(I) Determine which dataset or datasets are going to be used and to analyze their content.

(II) Apply preprocessing techniques with which non-numerical data is transformed into numerical and values are normalized with *zscore* or *mix-max* techniques.

(III) Evaluate the results obtained to determine if there are methods better than others, proposing therefore an efficient model of data preprocessing.

To develop this project, preprocessing analysis with classification deep neural networks, using different datasets will be proposed.

2. STATE OF THE ART

In this section a general approach of Intrusion Detection Systems, Machine Learning/Deep Learning and the combination of both is given.

2.1. INTRUSION DETECTION SYSTEMS HISTORY

First, a brief history of IDS is taken. It was with James Anderson's paper, *Computer Security Threat Monitoring and Surveillance* (1980), when intrusion detection analysis began. Since then, several studies and innovations have appeared. Anderson proposed that tracking data could be very useful to detect misuse and bizarre events.

In the following years, SRI International (Standford Research Institute) investigators developed different projects for the government. They created the first Detection Expert System (IDES) and later published *An intrusion detection model* where it was possible to find the necessary information to commercial intrusion detection system development, being this paper the basis for most of the future works. Meantime at the University of California, the researchers analyzed audit data by comparing it with defined patterns, stating this the beginning of Distributed Detection Systems (DIDS).

In the late '90s, the intrusion detection market gained popularity, mostly because Cisco recognized the importance of network intrusion detection.

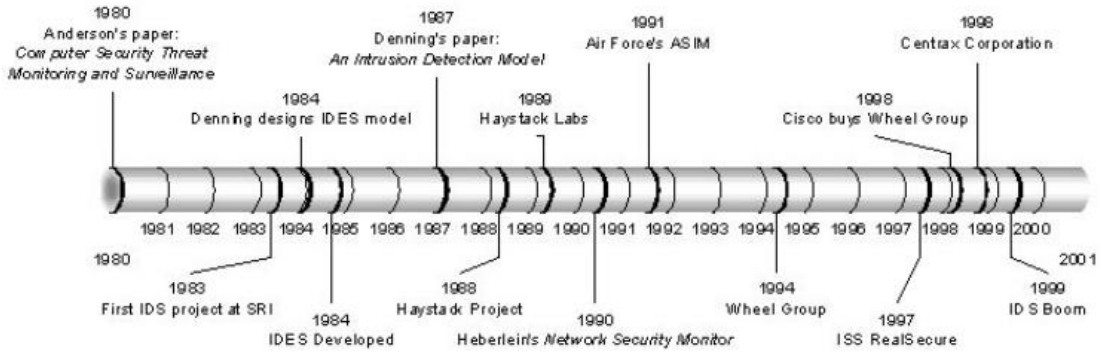

Figure 1. Evolution of IDS [1]

Nowadays it is possible to choose between different types of Intrusion Detection Systems [2] e.g. SIDS, NIDS or HIDS, and to difference between Anomaly Based or Signature Based.

2.2. ARTIFICIAL INTELLIGENCE

Artificial Intelligence is the second aspect of interest in this project and is introduced below.

The first recognized work involving artificial intelligence was back in 1943, by Warren McCulloch and Walter Pitts. They proposed a model of artificial neurons in which each neuron is characterized as being “on” or “off,” with a switch to “on” occurring in response to stimulation by a sufficient number of neighboring neurons. They showed, for example, that it is possible to compute any function by some network of connected neurons and most important, they suggested that suitably defined networks could learn by themselves. Later in 1950, two students built the first neural network computer, but it was Alan

Turing's vision the most influential. He published an article, *"Computing Machinery and Intelligence"*, where he introduced the Turing Test, named The Imitation Game at the beginning, machine learning, genetic algorithms, and reinforcement learning. The test consists in asking some questions to a computer and trying to find out if it is a human or a computer answering. From then to 1980 there have been multiple projects and researches although it was in the '80s when AI becomes an industry. The first commercial expert system, R1, began in 1982 at the Digital Equipment Corporation with a program that helped configure orders for new computer systems, saving the company over $40 million a year. A few years later, in 1986, the interest on neural network returned due to the investigations of at least four different groups that reinvented the back-propagation learning algorithm first found back in 1969 by *Bryson and Ho.*

Another important aspect faced since 2001 is that some experts stated that it might be more relevant to focus on the data rather than on the algorithm to apply. The issue is the availability of very large data sets, for example, trillions of words of English and billions of images from the Web *(Kilgarriff and Grefenstette, 2006)*; or billions of base pairs of genomic sequences *(Collins et al., 2003*).

With the background given, there is an extremely wide range of activities and subareas. These are the main applications [3, pp. 47-48].

- Robotic vehicles
- Speech recognition
- Image recognition
- Autonomous planning and scheduling
- Game playing
- Spam filtering
- Logistics planning
- Robotics
- Machine translation

The next step has been the incorporation of artificial intelligence in IDS, specifically with machine and deep learning techniques.

2.3. EXISTING DATASETS

In this section, the recent existing datasets to work with and to apply machine learning techniques are reviewed.

First datasets for IDS evaluation were known as DARPA datasets (*DARPA'98* and *DARPA'99* [4]) and were developed by the MIT Lincoln Lab. DARPA'98 collects 7 weeks of synthetic network traffic and 5M of connection records. The testing part chosen comprises just two weeks and 2M of connections. The result is 4GB of a compressed dataset. Otherwise, DARPA'99 span 5 weeks of network traffic although it is from different points – one inside the network and the other one outside-.

In the same year, 1999, built from the DARPA'98 dataset, *KDD Cup'99 dataset* was created by *Stolfo et al. (2000*). The dataset consists in 4,9M of single connection vector, each with 41 features, labeled either as normal or attack of one of the following categories: DoS, User to Root (U2R), Remote to Local (R2L) and Probing Attack. There are 24 training attack types, with an additional set of 14 different attacks that appear in the test data only, as said in [5]. KDD'99 is the most used IDS dataset in the latest years and there are multiple variations or improved KDD'99 datasets.

In 2009, Tavallae et al. [6] proposed the *NSL-KDD Dataset*, as an improvement over the *KDDCup'99*. The main advantages comparing to the original dataset are [6]:

- The classifiers will not have a tendency because the dataset does not include redundant records.
- Due to the number of records in the train and test sets, it is not necessary to select a small portion of the data to easily run the algorithm.

Among the most recent datasets, *ADFA-LD12, CTU-13, UNSW-NB15* and *UGR'16* are found.

ADFA-LD12, presented in *Creech and Hu (2013)* [7], focus on host-based intrusion detection systems evaluation. This dataset was released by the Australian Defence Force Academy. ADFA was generated on an Ubuntu Linux 11.04 host with Apache 2.2.17 and using the audit Unix tool to collect normal and attack Linux based system calls traces, for example, password brute-force against FTP and SSH or Java-based payloads. The traces can be classified into Training, Validation, Hydra-FTP, Hydra-SSH, Adduser, Java-Meterpreter, Meterpreter and webshell. The main inconvenient with the ADFA-LD12 is that it considers just one computer, with Linux OS.

CTU-13 dataset (Chez Technical University) is composed of 13 different malware captures, such as botnet content and normal and background traffic. This dataset is a good one despite some issues such as that there is not too much information about the network due to privacy reasons or that the authors just say that it is obtained from a university router. This dataset was described in [8].

UNSW-NB15, proposed by *Moustafa and Slay (2015)* [9]. Using IXIA PerfectStorm tool, in the Cyber Range Lab of the Australian Center for Cyber Security (ACCS), to generate automatic attacks so they can create nine families of real and updates attacks against several servers. The result is 2M of flows and 49 features for every flow. The problem with this dataset is the synthetic generation of traffic because it is more theoretical than realistic. This dataset differences between the following categories: Normal, Analysis, Backdoor, DoS, Exploits, Fuzzers, Generic, Reconnaissance, Shellcode and Worms.

UGR'16, described in [5] by *Macía-Férnandez et al. (2016)*, consists in a more than 4 months of netflow traces plus a set of realistic networking attacks specially designed to train and test IDS algorithms. The attacks implemented are DoS, Port Scanning and Botnet traffic. This dataset is valid for evaluating IDSs that considers long-term evolution and traffic periodicity. It can also be trained and evaluated by models that difference in daytime/nighttime or weekdays/weekends.

The features of the UGR'16 are timestamp of the end of a flow, duration of the flow, source IP address, destination IP address, source port, destination port, protocol, flags, forwarding status, type of service, packets exchanged in the flow and their corresponding number of bytes.

Table 1 contains a comparison of the different dataset mentioned for IDS evaluation in which the most important aspects of a dataset are exposed. First, there is a need to update both background and attack traffic, to avoid building a dataset that is not going to be valid in a few years time. Second, it is necessary to use realistic attacks, which has been done in the more recent dataset. However, providing realistic background traffic has not been done at the same effort level.

Table 1. Comparison of datasets for IDS evaluation

Dataset	Ref.	Real background traffic	Updated attack traffic
DARPA'98/'99	Darpa'98 and Darpa'99 Datasets	✗	✗
KDDCup99	KDD Cup'99 Dataset	✗	✗
NSL- KDD'09	NSL-KDD Dataset	✗	✗
CTU-13	CTU-13 Dataset	(✓)	✓
ADFA-2013	ADFA-2013 Dataset	(✓)	✓
UNSW-NB15	UNSW-NB15 Datastet	✗	✓
UGR'16	UGR'16 Dataset	✓	✓

✗: requirement not applied; ✓: requirement applied; (✓): requirement not completely applied or with certain limitations

2.4. PREPROCESSING

Preprocessing the information given in a dataset can be done in many ways and with very different results. It has been divided into three subtypes: **data reduction, missing-data treatment, and scaling.**

Data reduction is divided into feature selection and case selection. FS is the process of selecting a subset of features that have a significant or similar impact on the evaluation target as using all features. The purpose of FS was to increase the accuracy of the supervised learning algorithm although more intelligent techniques have been developed, classified as wrappers - to predict the benefits of adding or removing a feature, and as filters - to select features by evaluating independently prior to training. Most of the researches support the use of FS [10].

CS is similar to FS. It aims to identify and remove redundant and irrelevant cases. In this way, the size of the data set is reduced thus it is possible to save computing time and produce effective predictions [10]. These techniques are also applicable to FS. *Chirag et al., 2013* [11], used in their study packet preprocessing, removing redundant information from the given network packets.

Missing-data treatment can be divided into deletion (listwise deletion and pairwise deletion) or imputation (mean imputation, hot-deck imputation, cold-deck imputation, regression, etc.) [10].

Last but not least, scaling. Scaling can be defined as transforming the aspect of the entire dataset through different methods, **normalization** (normally between [0,1] or [-1,1]) and **Z-score standardization**. This treatment of the data transforms features according to a defined rule, so all the scaled features have the same degree of influence.

Standardization/normalization of datasets is a common requirement. They might behave incorrectly if the individual features do not more or less look like standard normally distributed data: Gaussian with zero mean and unit variance. In practice, the shape of the distribution is often ignored, and the data is just transformed to center it by removing the mean value of each feature, then scale it by dividing non-constant features by their standard deviation.

The elements usually used in a learning algorithm are normally centered around zero and have variance in the same order. When the variance of a feature is notably larger than the others it might dominate the objective function and make the algorithm unable to learn correctly.

To finish, different studies and projects are listed below, in relation to data preprocessing, between 2007 and 2018.

- *Li et al., 2007* [12], proposed an anomaly detection method based on TCM-KNN algorithm applied to pattern recognition, fraud detection, and outlier detection. As they say, it was the first time that machine learning algorithms are employed in intrusion detection, using an experimental dataset the KDD'99. Before beginning their experiments, they preprocessed the dataset. They use standardization in order to avoid one attribute dominating another.
  To evaluate their method, they used the two major indices of performance: the detection rate and the false positive rate, and training and testing a "clean" dataset (only normal instances) and an "unclean" dataset (normal and abnormal instances). The results showed that de detection performance vibrates just a little when using these two datasets.

- *Ma et al., 2008* [13] studied the impact of feature selection and data normalization on detecting anomaly network traffic, using KDD CUP 99 dataset and different machine learning algorithms (K-means, FCM and FE). They concluded that normalization is important due to the small but significant differences between the experiments done with normalized data and with not normalized data.

- *Wang et al., 2009* [14] evaluated the impact of different schemes of attribute normalization on the detection performance with three anomaly detection algorithms. They introduced four kinds of normalization: mean range [0, 1], statistical normalization, frequency normalization, and ordinal value normalization, also using KDD CUP 99 dataset. The techniques used to evaluate are KNN, PCA and SVM. This extended study concluded answering two questions: the importance of attribute normalization and what method of attribute normalization is most effective. For the first one, as in the other projects it is determined to be essential, but Wang et al. proved that if the data sample is large, statistical normalization and mean range [0,1] are the best option. Finally, they suggested that data can be scaled in a men range ([0,1] for example) to improve the detection performance. However, their experiments show that statistical normalization is the better choice if the data sample is large.

- *Kumar et al., 2014* [15] . Due to the symbolic or categorical attributes of the KDD cup99, they decided to preprocess before working with the data. First, they converted symbolic values to numeric values i.e. ICMP=1; TCP=2; UDP=3. After this conversion, they used the common Z-Score normalization.

- *Lokerwari et al., 2016* [16] also used KDD'99 dataset so they determined necessary to convert the nonnumeric attributes to numeric values, arbitrarily assigning values to each category (ICR:1; X11:2; …). Besides they applied min-max normalization to enclose the values to a specific range.

- *Chiba et al., 2018* [17], as the last two ones, considered converting KDD'99 nonnumeric attributes to numeric values by the same method of assigning values. They adopted normalization as well as a preprocessing technique, although they applied both, Z-Score and min-max. Among the different groups used, they concluded that the best results are obtained,

    - Using min-max, with 12 attributes
    - Using Z-Score, with 34 attributes

The table below shows the mentioned works.

Table 2. Preprocessing techniques in different projects

Reference	ML Approach	Dataset	Preprocessing technique
Yang Li et al., 2007	KNN	KDDCup'99	Z-Score
Wanli Ma et al., 2008	K-means, FCM and FE	KDDCup'99	Min-Max normalization
Wang et al., 2009	KNN, PCA and SVM	KDDCup'99	Min-Max normalization
Chirag et al., 2013	BC, AC and DT	-	Data reduction/missing-data
Kumar et al., 2014	ANN	KDDCup'99	Nonnumeric to numeric Z-Score
Huang et al., 2015	ANN, CART and CBR	ISBSG, Desharnais and USPFT	Data reduction/missing-data
Lokerwari et al., 2016	PSO WENN	KDDCup'99	Min-Max normalization Nonnumeric to numeric
Chiba et al., 2018	BPNN	KDDCup'99	Z-Score Nonnumeric to numeric Min-Max normalization

3. THEORETICAL FRAMEWORK

This section contains a general theoretical framework of the aspects related to this project.

3.1. INTRUSION DETECTION SYSTEM

Intrusion detection has a simple goal: to detect intrusions.

An intrusion detection system is a device or process that monitors systems and network activity for malicious, unauthorized activity or dangerous threats like malware, spyware, spam and many more. The main goal of IDS is to monitor and protect a system from possible misuses, attacks or compromised information, acting before there is real damage. Intrusion Detection Systems can be hardware, software or a combination of both.

Nowadays an IDS provides three main functions: monitoring, detecting and generating an alert. Any malicious activity is typically reported to an administrator or to a *security information and event management (SIEM)* system.

Intrusion detection systems can be classified in two: based on data source or based on the model of the intrusion. Each group can be divided in different types [18], as it can be seen in Figure 2.

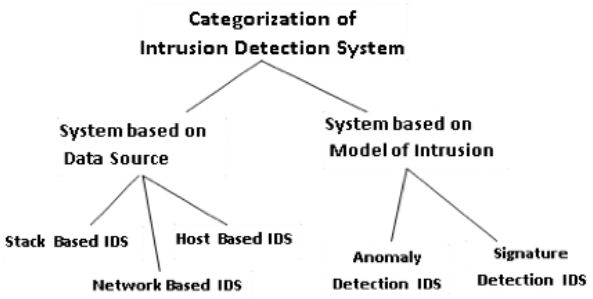

Figure 2. Categorization of IDS [18]

- **Stack Based Intrusion Detection Systems (SIDS)**

Stack based Intrusion Detection System (SIDS) is the latest technology, which works by integrating meticulously with the TCP/IP stack, allowing packets to be watched as they traverse their way up the OSI layers. Watching the packet in this way allows the IDS to pull the packet from the stack before the OS or application has a chance to process the packets.

- **Network Based Intrusion Detection System (NIDS)**

Network based Intrusion Detection Systems (NIDS) monitors the traffic as it flows to another host. Monitoring criteria for a specific host in the network can be increased or decreased with relative ease. NIDS should be capable of standing against large amount of network traffic to remain effective. As network traffic increases exponentially NIDS must grab all the traffic and analyze in a timely manner.

- **Host Based Intrusion Detection System (HIDS)**

Host based Intrusion Detection System (HIDS) keeps record of the traffic that is originated or is projected to originate on a particular host. HIDS controls the privileged access of the host to monitor specific components of a host that are not readily accessible to other systems. HIDS has limited view of entire network topology and it cannot detect attack that is targeted for a host in a network which does not have HIDS installed

- **Signature Based Intrusion Detection System**

Signature based Intrusion Detection System uses a set of rules to identify intrusions by watching for patterns of events, specifically, to known and documented attacks. It is typically connected to a large database which stocks attack signatures. These types of systems are able to detect only attacks to its database. Thus, if the database is not updated regularly, new attacks could slide through. Signature based IDS's affect performance when intrusion patterns match several attack signatures. In such cases, there is a noticeable performance lag. Signature definitions stored in the database need to be specific so that variations on known attacks are not missed. This can lead in building huge databases which eat up a chunk of space.

- **Anomaly Based Intrusion Detection System**

Anomaly based Intrusion Detection System examines ongoing traffic, activity, transactions and behavior in order to identify intrusions by detecting anomalies. It works on the notion that attack behavior differs enough from normal user behavior and it can be detected by cataloging and identifying the differences involved. The system administrator defines the baseline of normal behavior. Anomaly-based IDS systems are very prone to a lot of false positives and can cause heavy processing overheads on the computer system.

In this document, intrusion detection system based on the model of intrusion is going to be considered. Although signature detection has higher accuracy on result, this technique is less indicated to real-time application, this project is focused on anomaly detection methods.

3.2. MACHINE LEARNING AND DEEP LEARNING

It is necessary to briefly describe what Artificial Intelligence is. AI could be defined as the capacity given to computer systems to simulate human intelligence, meaning that to simulate processes such as learning, reasoning and self-correction. There are many definitions, approaching them from two different aspects such as behavior (acting), or thought processes and reasoning (thinking).

In the table below it can be observed some examples to understand the difference between acting and thinking.

Thinking Humanly	**Thinking Rationally**
"The exciting new effort to make computers think . . . *machines with minds*, in the full and literal sense." *(Haugeland, 1985)* "The automation of activities that we associate with human thinking, activities such as decision-making, problem solving, learning . . ." (Bellman, 1978)	"The study of mental faculties through the use of computational models." *(Charniak and McDermott, 1985)* "The study of the computations that make it possible to perceive, reason, and act." *(Winston, 1992)*
Acting Humanly	**Acting Rationally**
"The art of creating machines that per- form functions that require intelligence when performed by people." *(Kurzweil, 1990)* "The study of how to make computers do things at which, at the moment, people are better." *(Rich and Knight, 1991)*	"Computational Intelligence is the study of the design of intelligent agents." *(Poole et al., 1998*) "AI ...is concerned with intelligent behavior in artifacts." *(Nilsson, 1998)*

Figure 3. Differences between acting and thinking [3]

Machine learning is a combination of Computer Science and statistics that seeks to solve a problem on a computer by creating different algorithms [19]. The main goal of ML is to build systems that can be used by computers to automatically improve a specific task, using the training data or past experience.

Whereas Computer Science has focused primarily on how to manually program computers, Machine Learning focuses on the question of how to get computers to program themselves (from experience plus some initial structure). On the other hand, Statistics has focused primarily on what conclusions can be inferred from data. Additional questions appeared such as how to effectively capture, store, index, retrieve and merge these data; how multiple learning subtask can be orchestrated in a large system or just tractability questions.

Artificial intelligence includes different techniques, such as

- Decision tree
- Ruled based
- Data mining
    - Fuzzy logic
    - Genetic algorithms
- More Machine learning techniques

3.2.1. MACHINE LEARNING TECHNIQUES

Major machine learning techniques includes the following [2]:

NEURAL NETWORK (NN)

A neural network [20] is a set of simple units called neurons. A neuron is a linear automata, which realizes a weighted sum of several inputs according to a set of weights, and then computes a function, to obtain an output value. To interconnect all these neurons, the topology defines an *input layer*, where activations of the neurons are set to the input values, and an *output layer*, where the reading of the activations of the neurons gives the answer of the set.

Neural networks are composed [3, pp. 748-750] of nodes connected by directed links. A link from unit to unit serves to propagate the activation and also has a numeric weight, also called trainable parameter, associated with it, which determines the strength and sign of the connection. Then it applies an activation function, in which weights are summed and basically it governs the threshold at which the neuron is activated and the strength of the output signal. It is usual to use nonlinear functions - like the logistic function also called the *sigmoid function*, because it is possible to output a value between 0 and 1.

Single-layer and multi-layer neural networks are explained below.

SINGLE-LAYER NEURAL NETWORK

When all the inputs are connected directly to the outputs, it is called a single-layer neural network. All the computations are performed by the perceptron (a single neuron) so single-layer neural networks are also called *perceptron network*.

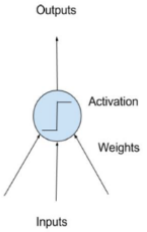

Figure 4. Simple NN

MULTI-LAYER NEURAL NETWORK (DEEP NEURAL NETWORK)

Multi-layer Neural Network or *Multi-layer Perceptron (MLP)* consists at least of three layers: input layer, hidden layer and output layer, as represented in Fig 3. The weights assigned to the connections are estimated using the Back Propagation [21] algorithm.

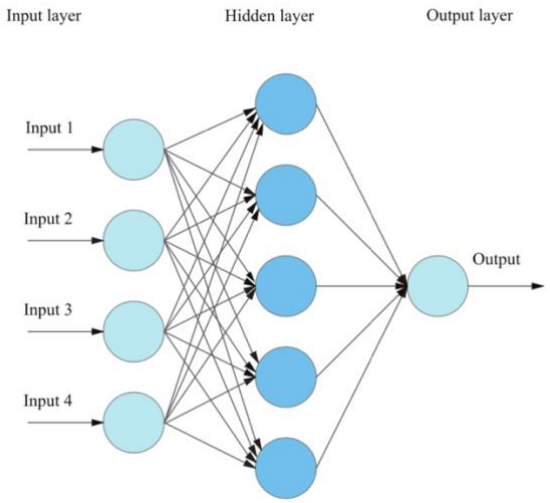

Figure 5. Simple MPL

The main advantage of NN is their tolerance to imprecise data and uncertain information, and their ability to infer solutions from data without having prior knowledge of the regularities in the data. This in combination with their ability to generalize from learned data has made them an appropriate approach to Intrusion Detection [22, p. 8].

BAYESIAN NETWORK

A Bayesian Network (BN) is a model that encodes probabilistic relationships among variables of interest. Combined with statistical schemes it is a technique used for intrusion detection. However, it has a major disadvantage that requires to make extra computation efforts to evaluate the results [22, p. 9].

MARKOV MODEL

There are two types of approaches [22, p. 9]:

Markov chains: A Markov chain is a set of states that are interconnected through certain transition probabilities, determining the topology and the capabilities of the model. It is used for anomaly detection, comparing the anomaly score obtained for the observed sequences with a fixed threshold.

Hidden Markov models: the system of interest is assumed to be a Markov process in which states and transitions are hidden.

Multiple studies have been presented with a common denominator: the systematic application of learning techniques to automatically obtain profiles of normal behavior.

SUPPORT VECTOR MACHINE

Support Vector Machine is a technique, as said in [22, p. 10], used for solving a variety of learning, classification and prediction problems. The basic SVM deals with two-class problems, in which data is separated by a hyper plane defined by a number of support vectors. Support vectors are a subset of a training data used to define the boundary between the two classes. Some applications in intrusion detection have been the extraction of features of the KDD dataset, for example.

In the figure below, there is a scheme including available machine learning algorithms.

Figure 6. Machine learning map

There is a wide variety of applications for machine learning nowadays, for example, speech recognition, computer vision, bio-surveillance, robot control or accelerating empirical sciences, although the main purpose in this document is machine learning applied to intrusion detection systems.

3.2.2. DEEP LEARNING

Deep learning is a sub-field of machine learning based on computational models of multiple layers to learn different levels of representation and abstraction, helping data to make sense so machine learning can move closer to Artificial Intelligence.

3.2.3. CLASSIFICATION IN MACHINE LEARNING

Machine Learning is divided in three types: supervised learning, unsupervised learning and semi-supervised learning [3, pp. 712-714].

Supervised learning is used to make the algorithm learn the mapping function from a given input-output pair, $y = f(x)$

Supervised learning offers two possible ways: classification or regression.

Regression model is the approach of approximating the mapping function from input values to *continuous (numerical)* outputs whereas *Classification* model is the approach of approximating the mapping function from an input to a *discrete (categorical)* output.

Some of the classifiers are:

- Naïve Bayes
- Support Vector Machine (SVM)

- K-Nearest Neigbour (KNN)
- Decision Tree
- Neural Network (NN)

Within the classifiers, it is possible to categorize *single classifiers* and *multiple classifiers.* Single classifiers are meant to classify only with two distinct classes or two possible outputs, e.g. male/female. Multiple classifier can classify with more than two classes, e.g. types of attacks.

In deep networks for supervised learning, which are intended to directly provide discriminative power for pattern classification purposes, target label data are always available in direct or indirect forms for such supervised learning. They are also called discriminative deep networks.

Unsupervised learning differs from the previous one in the fact that the output is up to the user, meaning that there are different ways to manage the data. It is no usually expected an output based on an input-output pair.

Deep networks for unsupervised learning are intended to capture high-order correlation of the observed or visible data, for pattern analysis or synthesis purposes when no information about target class labels is available.

These are some of the deep networks for unsupervised learning:

- Deep Belief Network (DBN)
- Boltzman Machine (BM)
- Restricted Boltzman Machine (RBM)
- Deep Neural Network (DNN)
- Deep autoencoder

In **semi-supervised learning**, there are different labeled examples so that it can reduce the training data required on the learning stage, being it a mix of both supervised and unsupervised

As mentioned before, the main focus is anomaly detection due to its capacity of detecting new and known attacks, although it increases the false alarm rate, because it can also detect normal packets as attacks and vice versa. Having large quantities of information of TCP/IP packets it is possible to develop powerful algorithms to improve the false alarm rate. Thanks to machine learning it has been possible to open a new area, deep learning, to define complex models. According to [23], deep learning algorithms have witnessed as a powerful algorithm due to the remarkable results in speech processing and natural language. The primary reason to that are that the deep learning algorithms have been associated with the two most important characteristics:

- hierarchical feature representations

- learning long-term dependencies of temporal pattern in large scale sequence data.

There are two phases in every machine learning algorithm: **training and testing.** To train the dataset it is highly recommended to prepare the information given as the input. This preparation is called data preprocessing. It includes data cleaning, normalization, transformation, feature extraction and selection, etc. As a result of this process the final training data is obtained.

3.3. DATA PREPROCESSING

As preprocessing is the main goal of this project, in this section different preprocessing techniques are going to be described.

There are many aspects on preprocessing that can have a positive impact on the result of the algorithm [24], such as instance selection and outlier detection, processing unknow features values, choosing the interval borders and the correct arity for the discretization, feature selection methods or data normalization techniques.

A dataset [25] is the collection of labeled or not labeled examples in which each element is called **feature vector** and it contains different values. The problem transforming the data of the dataset is called **feature engineering** and depending on the technique applied, the results are improved.

In the following part the different techniques used are described [25].

3.3.1. ONE-HOT ENCODING

Most of the learning algorithms only work with numerical features. If the dataset used have categorical features such as "type of flower" or "colors", it is highly recommended to transform a categorical one into several binary ones.

First option is to *encode the text to dummy variables*. If you have a categorical feature "colors" with three possible values: "red", "green" and "blue", you can transform the feature into a vector of three numerical values:

red = [1,0,0]
green = [0,1,0]
blue = [0,0,1]

Second option is to *encode the text values to indexes*. Using the same example, you can transform into indexes:

red = [1]
green = [2]
blue = [3]

3.3.2. NORMALIZATION

Normalization is a "scaling down" transformation of the features. Normally there is a large contrast between the maximum and the minimum values so normalizing the features minimize the complexity involved on handling all the data.

According to [17], normalization is mainly beneficial for classification algorithms relating neural networks. Moreover, if back-propagation is used in the NN, normalizing the input values will help to accelerate the learning phase, consequently becoming a more efficient neural network.

MIN-MAX NORMALIZATION

Min-Max Normalization is converting the actual range into a standard range of values, typically in the interval [-1,1] or [0,1].

The normalization formula looks generally like in (1):

$$x'^{(j)} = \frac{x^{(j)} - min^{(j)}}{max^{(j)} - min^{(j)}} \quad (1)$$

where $min^{(j)}$ and $max^{(j)}$ are, respectively, the minimum and the maximum value of the feature j in the dataset

STANDARDIZATION

Standardization or z-score normalization is scaling the features so that they have the properties of a standard normal distribution with $\mu = 0$ and $\sigma = 1$, being μ the mean (the average value of the feature, averaged over all examples in the dataset) and σ the standard deviation from the mean.

Standard scores are calculated as follows (2):

$$x'^{(j)} = \frac{x^{(j)} - \mu^{(j)}}{\sigma^{(j)}} \quad (2)$$

Depending on the learning algorithm or the type of data is being managed, it is advisable to apply Min-Max Normalization or Z-Score.

Normally, it could be generalized as follows:

- if using unsupervised learning, it is usual to benefit from *standardization*;
- if features are close to a normal distribution or the dataset has outliers (extremely high or low values), *standardization* is preferred;
- for all the other cases, *normalization* is recommended.

3.3.3. DEALING WITH MISSING FEATURES

Dealing with missing features techniques include:

- if the dataset is big enough so you can sacrifice some example, the easiest way is removing the example/s with missing features;
- using a learning algorithm capable of dealing with missing features;
- using data imputation technique, that consist on replacing the missing value with a feature obtained by an average value.

3.4. DEEP LEARNING ALGORITHM

3.4.1. ARCHITECTURE OF A NEURAL NETWORK

As it has been mentioned before, Deep Learning Neural Network is a Neural Network with many hidden layers. Basically, neural networks receive an input, called feature vector, and produce an output. In a **Classification Neural Network,** it is expected a class/category prediction/output, whereas in Regression Neural Network it is expected a single prediction.

The diagram bellow shows a typical neural network.

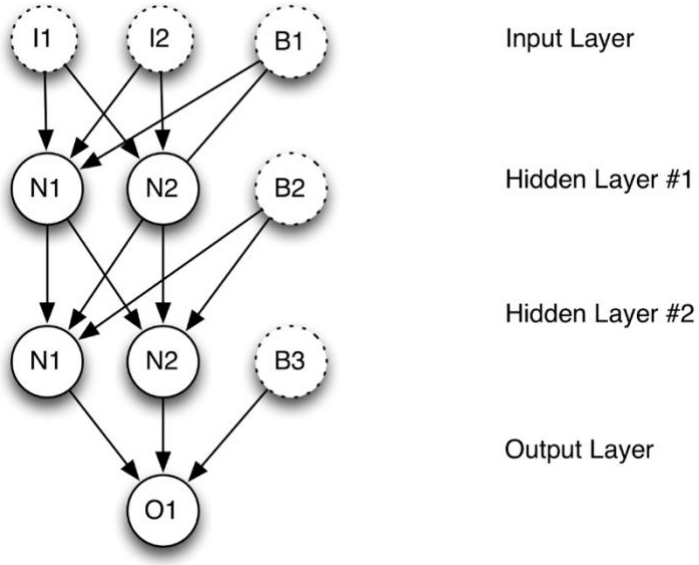

Figure 7. Scheme of a typical deep neural network

There are four kind of neurons, grouped into layers. **Input neurons** are mapped to one element in feature vector, **hidden neurons** allows to process the input into the output, **output neurons** calculate individually a part of the output, c**ontext neurons** maintain the connection between calls to the neural network and **bias neurons** store the value of 1, to make possible to move the activation function left or right on the graph. It is usual to have a bias neuron in the output layer and hidden layers.

To calculate the output of each layer, and so the final output, **activation functions** are used. Adding layers to a neural network, increases the ability to learn complex non-linear patterns, so that modern deep networks use nonlinear activation functions *Softmax* - to output the final classification, and *ReLU*- to output the hidden layers. *Linear* activation function is used for the output of regression networks or 2-class classification.

3.4.2. TRAINING AND VALIDATION

The goal of the model built is not just to make good predictions with data already seen or trained, but to be good at predicting with new data. Due to that, when training a network, the original dataset is separated in various datasets: **training set, validation set** and **test set.**

The training set, usually the biggest one, is used to build and train the model. The validation set and training set, also called **holdout sets,** are used to choose the learning algorithm and to find the best values of hyperparameters, so they cannot be used to build the model whereas they affect to the capacity and to the learning characteristics of the model.

On the other hand, **early stopping** is a technique used to prevent overfitting. Overfitting appears when the trained network begins to memorize rather than generalize. Early stopping callback- a callback provide a way to execute code and interact with the training model process, permits to specify the performance measure to monitor in order to end the training [26].

3.4.3. MODEL PERFORMANCE ASSESSMENT

CLASSIFICATION ACCURACY

Classification Accuracy, usually called just Accuracy, is the number of correct predictions made as a ratio of all predictions made, being True Positives and True Negatives the correct predictions, and the True Positives, True Negatives, False Positives and False Negatives all the predictions made, as it can be seen in (3).

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions\ made} = \frac{TP + TN}{TP + FP + TN + FN} \quad (3)$$

CONFUSION MATRIX

Confusion Matrix, also called error matrix, allows the visualization of the performance of an algorithm. The table represents predictions on the x-axis and real values on the y-axis. A 2-classes neural network would have the following confusion matrix.

Confusion Matrix

True label	Class 1 - positive	True Positive	False Negative
	Class 2 - negative	False Positive	True Negative
		Class 1 - positive	Class 2 - negative
		Predicted label	

To rapid evaluate the confusion matrix there must be a clear diagonal so it can be appreciated that the true label has been predicted properly.

It is possible to calculate the confusion matrix accuracy as the True Positives plus the False negatives, divided for the total samples used (4).

$$Accuracy = \frac{True\ Positives + False\ Negatives}{Total\ number\ of\ samples} \quad (4)$$

PRECISION AND RECALL

Precision-Recall is a measure of prediction useful when the classes are imbalanced. Precision measures the relevancy meanwhile recall measures how many truly relevant results are returned.

It is possible to plot the precision-recall curve that shows a compensation between precision and recall for different threshold. A high area under the curve represents both high values, meaning, for precision, a low false positive rate; and for recall, a low false positive rate. With all that, it can be concluded that high scores for both prove accurate results.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \quad (5)$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \quad (6)$$

Besides, precision and recall are related with the **F_1 score**, defined as the harmonic mean of precision and recall.

$$F1 = 2\ x\ \frac{Precision\ x\ Recall}{Precision + Recall} \quad (7)$$

AREA UNDER THE ROC CURVE

Area Under the ROC Curve, or AUC to simplify, represents the ability of the algorithm to discriminate between positive and negative classes. It is usually used for binary (2-classes) problems.

AUC is the area under the curve of plot False Positive Rate vs True Positive Rate, being

- **True Positive Rate** or **Sensitivity** is the number of instances from the positive class that have been predicted correctly.

$$TPR = \frac{True\ Positive}{False\ Negative + True\ Positive} \quad (8)$$

- **False Positive Rate** or **Specificity** is the number of instances from the negative class that have been predicted correctly.

$$FPR = \frac{False\ Positive}{False\ Positive + True\ Negative} \quad (9)$$

3.4.4. TECHNOLOGIES USED

TENSORFLOW AND KERAS

TensorFlow is an open source software library for ML. Originally developed by the Google Brain team and actually maintained and used internally by Google, TensorFlow permits both researchers and developers, to easily investigate, build and deploy ML applications. The main advantage of this framework is that it is possible to run deep learning models on a local CPU, cloud CPU or GPU and even Android devices.

TensorFlow is a low-level mathematics API built for deep learning. There are two possible procedures to use it: directly and with Keras.

Keras is a layer on top of TensorFlow widely adopted by deep learning engineers. This framework allows to easily create a neural network, by defining the individual layers in a much more high-level API, so its simplicity and readability makes Keras a good alternative.

CONDA AND JUPYTER

Conda [32] is an open source package management system and environment management system that runs on Windows, macOS and Linux. It quickly installs, runs and updates packages and their dependencies. It was created for Python programs, but it can package and distribute software for any language. It is one of the most popular managers for Python users, offering the possibility of setting up, switching and deleting environments.

Conda has been used to create an optimal environment to develop the project. In this environment, we have also used the Jupyter Notebook. The Jupyter Notebook [33] is an open-source web application that allows creating and sharing documents that contain live code, equations, visualizations and narrative text. It has been very useful for graphics visualization.

PYTHON LIBRARIES

NUMPY

Numpy [27] is the fundamental package for scientific computing with Python. It allows to work with N-dimensional array objects, tools for integrating C/C++ and Fortran Code and useful linear algebra, Fourier transform, and random number capabilities, among other things.

SCIPY

Scipy [28] is a Python-based ecosystem of open-source software for mathematics, science and engineering. It is fundamentally built on a NumPy N-dimensional array package

MATPLOTLIB

Matplotlib [29] is a Python 2D plotting library which produces quality figures in a variety of formats and interactive environments across platforms.

SCIKIT-LEARN

Scikit-learn [30] is a simple and efficient ensemble of tools for data mining and data analysis, accessible and reusable, built on NumPy, SciPy and matplotlib.

We have paid special attention in this project to the following tools provided by Scikit-learn:

- Metrics
- Preprocessing
- Models selection

PANDAS

Pandas [31] is an open source library that provides high-performance, easy-to-use data structures data analysis tools for the Python programming language.

The library has been used to load and work with datasets.

GOOGLE COMPUTE ENGINE

Google Compute Engine [34] is the Infrastructure as a Service (IaaS) component of Google Cloud Platform. It enables users to launch virtual machines (VMs) on demand.

In this project, this platform has been used to set up a deep learning environment, creating multiple VMs and making it possible to work with machines with high capabilities.

4. CASE STUDY

In this section, the development of the main project is related.

4.1. DATASETS USED

In the following part, the datasets used for the case study will be described in depth. UGR16, UNSW-NB15 and NSL-KDD are the datasets chosen to accomplish the project. The election has been based, on the one hand, on the fact that the first one has real background traffic and updated traffic, the second one just updated traffic and the last one none of them, having therefore a large variety of traffic, and on the other, because UGR16 is a relatively new Spanish dataset, and NSL-KDD and UNSW-NB15 are recent datasets commonly used for intrusion detection tests based on machine learning algorithms as mentioned in related works.

There has been special interest in the features that form these datasets, the type of the future (non-numerical/object or numerical), the percentages for attacks classification and the dataset size. Special attention has been made to the final feature that gives us the information that determines o if it is an attack or not and to the nature of each feature so as to know the best way to work. This last mentioned will be detailed in next sections.

Appendix A contains detailed tables with all the possible values for each feature in each dataset.

4.1.1. UGR16

The UGR16 dataset has 13 features: timestamp of the end of a flow (*time*), dura- tion of flow (*duration*), source IP address (*sip*), destination IP (*dip*), source port (*source_port*), destination port (*dest_port*), protocol (*protocol*), flags (*flags*), forwarding status (*forward_status*), type of service (*type_service*), packets exchanged in the flow (*pack_exchanged*), their corresponding number of bytes (*bytes*) and the type of attack (*attack_tag*).

In this case, it has been found a multiclass problem and *attack_tag* can take the following values: bakcground, anomaly-spam, blacklist, DoS, scan11, scan44, botnet, UDP scan and SSH scan, considering background as normal traffic.

The table below shows the features and its nature, indicating (O) that it is an object value and if not, that is it a numeric value. As it can be observed, *attack_tag* is not considered in the table. However, it is an object feature.

Table 3. UGR16 features

time
duration
sip (O)
dip (O)
source port
dest port
protocol (O)
flags (O)
forward status
type service
pack_exanged
bytes

(O): Object type feature

There are multiple options to get the dataset. August week 2, test version, has been selected to work with, due to the fact that the test's versions have synthetic traffic so more varied data can be worked with. The original version of the dataset is 81GB, but for this project it has been selected a 1.4GB portion to work with, in which there were found the attacks value in Table 4.

Table 4. Categories found in the attack_tag feature in the UGR16

Name	Percentage (%)
Background	87,18
Other attacks	12,82

The *time* feature has been deleted for this project since the evaluation of the IDS as model based on time series is not in the scope of the present document. Finally, the project has been carried out using 12 features when using UGR16.

As just mentioned, a relatively big portion of data has been used, ins comparison with the other two datasets exposed below. Owing to the size of the working data, it has been impossible to develop the project and process the data needed in our personal computer due to computing characteristics- one processing unit of 2,3GHz with 2 cores, 8GBof RAM and 70Gb of free SSD. Consequently, a cloud computing as remote solution has been employed, particularly, Google Cloud Platform services. The characteristics of the VMs have to be customized and for this project they are set as: 8 vCPUS (Virtual Central Processing Units), 30 GB of RAM and 1TB of SSD.

4.1.2. NSL-KDD

NSL-KDD dataset has 42 features, classified in 3 categories: **basic features, content features** and **traffic features.**

- Basic features include all the attributes that can be extracted from an individual TCP/IP connection.

- Content features consist of some concrete features needed to detect attacks that show suspicious behavior in the data portion, eg., number of failed login attempts.
- Traffic features include the features computed with respect to a window interval.

The table below contains the classified features of the NSL-KDD.

Table 5. NSL-KDD features

Basic features	Content features	Traffic features
duration	hot	count
protocol_type (O)	num_failed_logins (O)	srv_count
service (O)	logged_in	serror_rate
flag (O)	num_compromised	srv_serror_rate
src_bytes	root_shell	rerror_rate
dst_bytes	su_attempted	srv_rerror_rate
land (O)	num_root	same_srv_rate
wrong_fragment	num_file_creations	diff_srv_rate
urgent	num_shells	srv_diff_host_rate
	num_access_files	dst_host_count
	num_outbound_cmds	dst_host_srv_count
	is_host_login (O)	dst_host_same_srv_rate
	is_guest_login (O)	dst_host_diff_srv_rate
		dst_host_same_src_port_rate
		dst_host_srv_diff_host_rate
		dst_host_serror_rat
		dst_host_srv_serror_rate
		dst_host_rerror_rate
		dst_host_srv_rerror_rate

(O): Object type feature

As with the UGR16, it has been dealt with a multiclass problem. The final feature is *attack_tag* and it has not been take into account either in the table although it is an object value too, in which it is differenced: normal, neptune, satan, ipsweep, portsweep, smurf, nmap, back, teardrop, warezclient, pod, guess_passwd, buffer_overflow, warezmaster, land, imap, rootkit, ladmodule, ftp_wirte, multihop, phf, perl and spy, being normal considered as normal traffic.

This dataset, training version, is 19.1MB so it has not been necessary to select a portion. In Table 6 the percentages of each attack class can be found.

Table 6. Categories found in the attack_tag feature in the NSL-KDD

Name	Percentage (%)
Normal	53,56
Other attacks	46,54

To work with this dataset, considering its reduced size, it has not been indispensable to use Google's VMs the decision was to developpe the model in local, frequently using Jupyter Notebook.

4.1.3. UNSW-NB15

UNSW-NB15 dataset has 49 features classified in 5 categories: **flow features**, **basic features**, **content features**, **time features** and **additional generated features**.

- Flow features include the identifier attributes between hosts.
- Basic features, unlike in the NSL-KDD, include the attributes that represent protocol connections, most of them similar to basic features in NSL-KDD.
- Content features involve the attributes of TCP/IP and some http connections.
- Time features contains all the attributes related with time, e.g., arrival time between packets.
- Additional generated features divided in two groups: General purpose in order to protect the service of protocols and Connection features.

Contrary to the previous two datasets, UNSW-NB15 has an *attack*_tag feature although the final category, *Label*, is binary, indicating attack or no attack. Since our proposal involves dealing with multiclassification problems, it was mandatory to modify the UNSW-NB15 so it could be treated as it as multiclass classification. This will be explained forward. However, both *attack_tag* and *Label* are object features and the possible values for the first one mentioned are: normal, generic, exploits, fuzzers, DoS, reconnaissance, analysis, backdoor, shellcode and worms.

It has been chosen the training set, that only has 45 features- just 1 feature classified as flow features, and it is 15.4MB. The following tables show the features and the percentages of attacks classifications used.

Table 7. UNSW-NB15 features

Basic features	Time features	Content features	Additional gen. Features
dur (O)	sintpkt	swin	ct_srv_src
proto (O)	dintpkt	stcpb	ct_state_ttl
service (O)	djit	dtcpb	ct_dst_ltm
state	sjit	dwin	ct_src_dport_ltm
spkts	tcprtt	smean	ct_src_sport_ltm
dpkts	synack	dmean	ct_dst_src_ltm
sbytes	ackdat	trans_depth	is_ftp_login
dbytes		response	ct_ftp_cmd
rate			ct_flw_http_mthd
sttl			ct_src_ltm
dttl			ct_srv_dst
sload			is_sm_ips_ports
dload			
sloss			
dloss			

(O): Object type feature

Table 8. Categories found in the Label feature in the UNSW-NB15

Type of attack	Percentage (%)
No attack	55,06
Attack	44,94

In the same way as in the previous dataset, the UNSW-NB15 has been processed in local, using Jupyter Network occasionally.

4.2. ARCHITECTURE OF THE NETWORK

This project follows a Classification Neural Network and its characteristics are described below.

First step has been setting up the layers. To do so, 3 hidden layers have been used, input and output layer, using activation function *ReLU* for the hidden and input ones, and *Softmax* for the output one.

Second, the model is compiled using the following settings:

- *Loss function*: categorical_crossenntropy, since a multiclass problem-2 or more output classes, is the aim of the proposed project.
- *Optimizer:* adam, due to its popularity in the field of deep learning because it obtains good results fast.
- *Metrics:* accuracy because it is one of the most popular metrics.

Third, the model is trained with the previously split sets, 25% test and 75% validation, using *EarlyStopping*.

Last, accuracy is evaluated and, therefore, predictions are made.

A summarizing table with the network characteristics and the model used is shown below [35].

Table 9. Neural network characteristics

Activation function	Input and hidden layers	*ReLU*
	Output layer	*Softmax*
Loss function	*Categorical_crossentropy*	
Optimizer	*adam*	
Metrics	*accuracy*	
Split test	Test	25%
	Validation	75%

```
# Break into X (predictors) & y (prediction)
x, y = to_xy(df,'Label')

# Create a test/train split.  25% test
# Split into train/test
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.25, random_state=42)

# Create neural network
model = Sequential()
model.add(Dense(10, input_dim=x.shape[1], kernel_initializer='normal',
activation='relu'))
model.add(Dense(50, input_dim=x.shape[1], kernel_initializer='normal',
activation='relu'))
model.add(Dense(10, input_dim=x.shape[1], kernel_initializer='normal',
activation='relu'))
model.add(Dense(1, kernel_initializer='normal'))
```

```
model.add(Dense(y.shape[1],activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-4, patience=5,
verbose=1, mode='auto')
history =
model.fit(x_train,y_train,validation_data=(x_test,y_test),callbacks=[monito
r],verbose=2,epochs=1000)

# Measure accuracy
pred = model.predict(x_test)
pred = np.argmax(pred,axis=1)
y_eval = np.argmax(y_test,axis=1)
score = metrics.accuracy_score(y_eval, pred)
print("Validation score: {}".format(score))
```

4.3. BEFORE PREPROCESSING

Although preprocessing is the main goal of this project, the data was prepared before preprocessing it.

4.3.1. ADDING HEADERS

The fundamental task has been adding a header to the dataset since the original datasets are not labeled. Thanks to this it has been possible to work with all the labeled features.

4.3.2. DROPPING COLUMNS

Last procedure before preprocessing has been dropping the columns/features that are not interesting for the project. It has been applied the *pandas.DataFrame.drop* [36] function.

As already mentioned, UGR16's *time* feature does not provide information of interest for the project, so it has been deleted.

NSL-KDD has a final *id* feature that does not give any information and it has been also dropped.

For UNSW-NB15, it has been necessary to transform it so it could be possible to work as if it was a multiclass problem. For achieving it, last column, *Label*, has been dropped and it has been used as output the *attack_tag* column.

4.4. PREPROCESSING METHODS APPLIED

Preprocessing has been applied before building the network previously mentioned.

As it was pointed out in the sections above, it has been necessary to analyze each dataset to know the nature of the features so as to decide which preprocessing technique to apply. In tables 3, 5 and 7 it is possible to see the features nature, considering object type both text and features with punctuation marks e.g., IP addresses.

Preprocessing in this project has been divided into two fundamental categories: **one-hot encoding** for object features and **normalization** for numerical features.

For the first one mentioned the following options are available: transforming to *indexes* or transforming to *dummy variables*. The method of "cleaning" features with punctuation marks has also

been included in this category. For the second one it has been considered encoding the values with *zscore* and *minmax*- between (0,1) and (-1,1).”

4.4.1. ONE-HOT ENCODING FOR OBJECT FEATURES

Since the NN only admits numeric values, it is essential to always encode all the non-numeric features. In the following sections the proposed methods for object features preprocessing are presented.

TEXT TO DUMMY

The function that has been applied can be observed in the following code lines, and to do so Pandas function *get_dummies* [37] has been used, converting categorical data into indicator variables.

```
# Encode text values to dummy variables
def encode_text_dummy(df, name):
    dummies = pd.get_dummies(df[name])
    for x in dummies.columns:
        dummy_name = f"{name}-{x}"
        df[dummy_name] = dummies[x]
    df.drop(name, axis=1, inplace=True)
```

All the object features have been encoded using dummy transformation because is the best option when there is a considerable quantity of options for each object feature, except the final feature, due to the different values for that feature are less.

For the UGR16, the columns encoded have been *protocol* and *flags.*

For the NSL-KDD, *protocol_type*, *service*, *flag*, *land*, *num_failed_logins*, *is_host_login* and *is_guest_login.*

For the UNSW-NB15, *dur*, *porto*, *service* and *attack_tag.*

TEXT TO INDEX

To encode labels with values between 0 and n-classes-1, the function below has been used applying *sklearn.preprocessing.LabelEncoder* class [3].

```
# Encode text values to indexes
def encode_text_index(df, name):
    le = preprocessing.LabelEncoder()
    df[name] = le.fit_transform(df[name])
    return le.classes_
```

To encode the final feature of each dataset this method has been used. Particularly, *attack_tag* for UGR16 and NSL-KDD and UNSW-NB15.

DATA CLEANING

It has been seen as mandatory to clean the source and destination IP addresses of the UGR16 due to its format contains dots e.g., 127.404.60.89, and without transformation it cannot be processed by the network.

The code used for this purpose is:

```
# Clean IP address
def clean_ip(s):
    s = ''.join([i for i in s if i not in frozenset(string.punctuation)])
    s_int = int(s)
    return s_int
```

The following table summarizes the object to numeric transformations made over the three datasets.

Table 10. Encoded features and method applied

Method	UGR16	NSL-KDD	UNSW-NB15
Text dummy	protocol flags	protol_type service flag land num_failed_login is_host_login is_guest_login	dur proto service attack_tag
Text index	attack_tag	attack_tag	attack_tag
Data cleaning	sip dip	-	-

4.4.2. NORMALIZATION FOR NUMERICAL FEATURES

Once all the features are numeric it is time to apply normalization and get results.

As explained in the theoretical framework, *MinMax Normalization* and Standard Normalization- or *Zscore*, have been considered. In the following paragraphs the functions used for each normalization are defined.

MINMAX NORMALIZATION BETWEEN 0 AND 1

```
# MINMAX (0,1)
# Encode a column to a range between normalized_low and normalized_high
def min_max_0(df, name, normalized_low=0, normalized_high=1,data_low=None,
data_high=None):
    if data_low is None:
        data_low = min(df[name])
        data_high = max(df[name])

     df[name] = ((df[name] - data_low) / (data_high - data_low)) \
          * (normalized_high - normalized_low) + normalized_low
```

MINMAX NORMALIZATION BETWEEN -1 AND 1

```
# MINMAX (-1,1)
# Encode a column to a range between normalized_low and normalized_high
def min_max_1(df, name, normalized_low=-1, normalized_high=1,data_low=None,
data_high=None):
    if data_low is None:
        data_low = min(df[name])
```

```
        data_high = max(df[name])

    df[name] = ((df[name] - data_low) / (data_high - data_low)) \
         * (normalized_high - normalized_low) + normalized_low
```

ZSCORE NORMALIZATION

```
# Encode a numeric column as zscores
def encode_numeric_zscore(df, name, mean=None, sd=None):
    if mean is None:
        mean = df[name].mean()
    if sd is None:
        sd = df[name].std()
    df[name] = (df[name] - mean) / sd
```

For data normalization **two approaches** have been followed:

- Applying the same method to all the features.
- Make try and failure trainings combining the two main methods, zscore and minmax, or none.

The first strategy, as previously mentioned, is to apply the same function- zscore, minmax between 0-1 and minmax between -1-1, to all the features. This have been made with the three datasets.

The second strategy has been implemented in two different ways, depending on the dataset used. As it has been explained before, NSL-KDD and UNSW-NB15 features can be classified in 3 and 4 groups respectively, whereas UGR16 features are not grouped due to there are considerably less.

Considering these characteristics, the decision has been to make variations rotating the normalization method in function of the category of feature, for the NSL-KDD and UNSW-NB15 datasets.

For the UGR16 the following methods have been combined:

Table 11. Variations over UGR16

Label	Var1_1	Var2_1	Var3_1
sip	zscore	0minmax	1minmax
dip	zscore	0minmax	1minmax
source port	-	-	-
dest port	-	-	-
forward status	-	-	-
type service	-	-	-
pack_exanged	-	-	-
bytes	-	-	-

Table 12 and table 13 summarize the variations made for the NSL-KDD and the UNSW-NB15

Table 12. Variations over NSL-KDD

Nº variation	Basic features	Content features	Traffic features
Var1_2	min_max_0	zscore	zscore
Var2_2	zscore	min_max_0	zscore
Var3_2	zscore	zscore	min_max_0
Var4_2	min_max_0	zscore	min_max_0
Var5_2	min_max_0	min_max_0	zscore
Var6_2	zscore	min_max_0	min_max_0
Var7_2	zscore	-	-
Var8_2	-	zscore	-
Var9_2	-	-	zscore
Var10_2	zscore	-	zscore
Var11_2	zscore	zscore	-
Var12_2	-	zscore	zscore

Table 13. Variations over UNSW-NB15

Nº variation	Basic features	Time features	Content features	Add. Features
Var1_3	zscore	min_max_0	min_max_0	min_max_0
Var2_3	zscore	-	-	-
Var3_3	zscore	zscore	-	-
Var4_3	zscore	zscore	zscore	-
Var5_3	-	-	-	zscore
Var6_3	-	-	zscore	-
Var7_3	-	zscore	-	-
Var8_3	-	zscore	zscore	-
Var9_3	-	zscore	zscore	zscore

4.5. MODEL EVALUATION AND RESULTS

As commented before, the object features have been encoded on all the occasions, so this section is just considering the performances made on numerical features.

The results have been measured, on the one hand, with the validation score given by the network and its correspondent accuracy graphic, in which it is observable the evolution of the training and evaluation sets. On the other, the confusion matrix that has been obtained, which offers a simple comparison between the input features and the predicted ones. These two methods are the basic performance assessments for multiclassification problems.

Before beginning with the different approaches, a *zero approach* has been done in which the model has been evaluated, with the three datasets, without any encoding, except object features as previously explained. For this practice, the following results have been collected:

UGR16

Option 1. Encoding just object values, including IP addresses transformation to numerical values, the validation score was 0,00701.

Option 2. Encoding just object values but removing source and destination IP addresses features, the validation score obtained a 0,87685.

NSL-KDD

Encoding just the object labels the validation score was 0,9501.

UNSW-NB15

In the same way as in the previous dataset, the validation score was 0,55803

In the figure below the four results and its correspondent accuracy graphic and confusion matrix are offered:

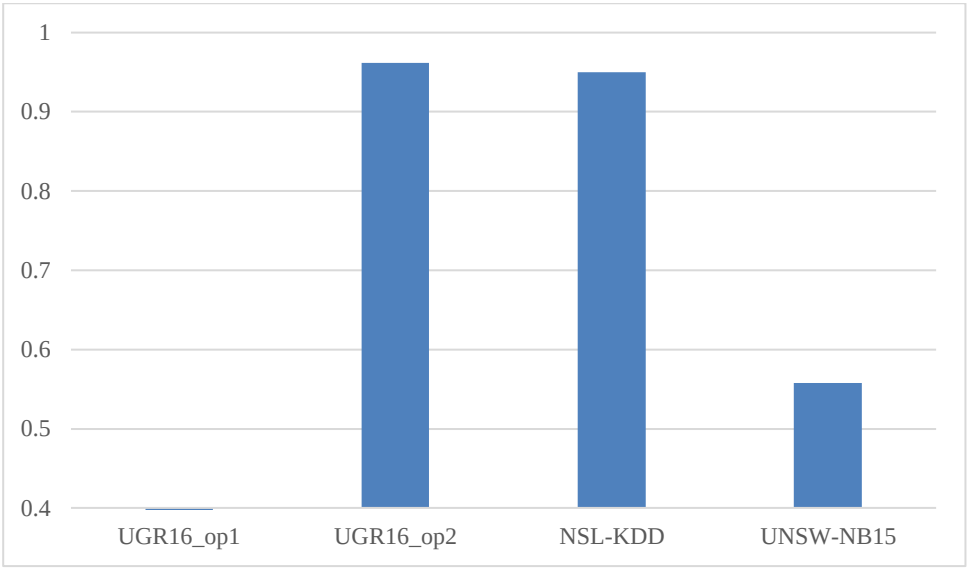

Figure 8. Validation scores for Approach 0

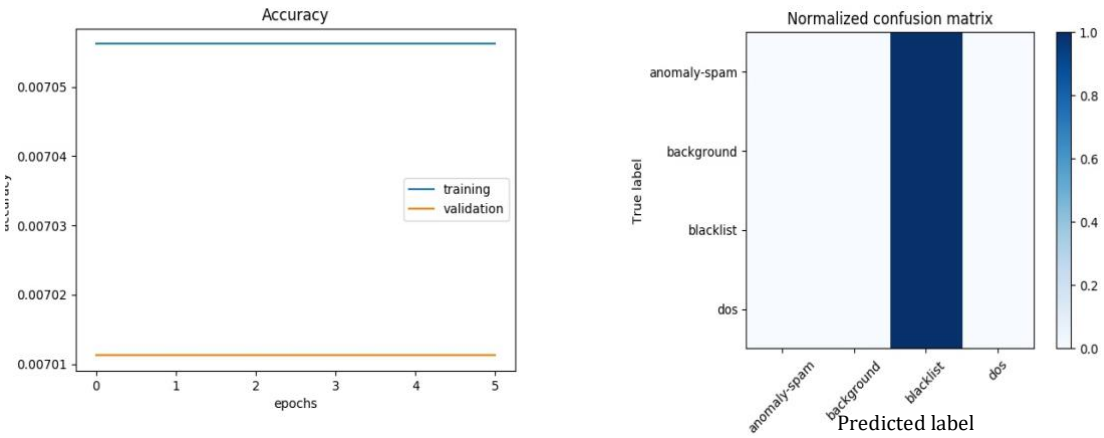

Figure 9. UGR16 approach 0_op1 accuracy graphic and confusion matrix

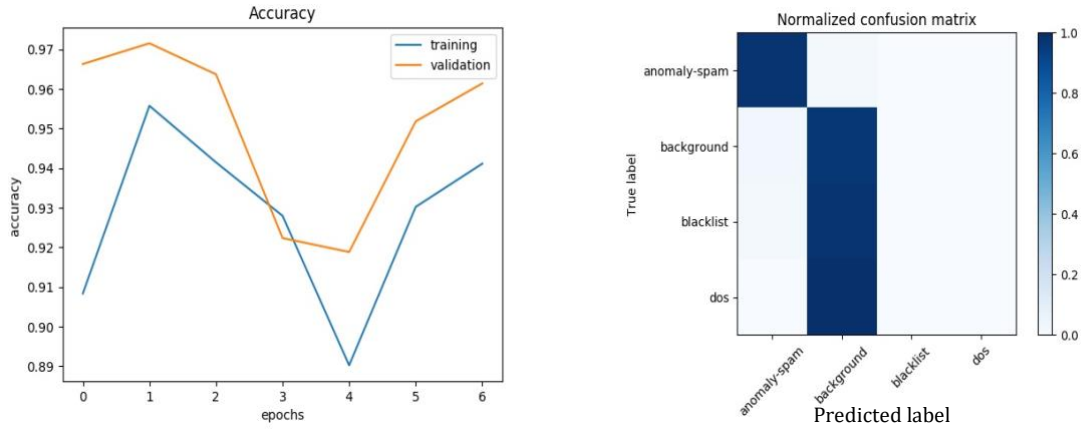

Figure 10. UGR16 approach 0_op2 accuracy graphic and confusion matrix

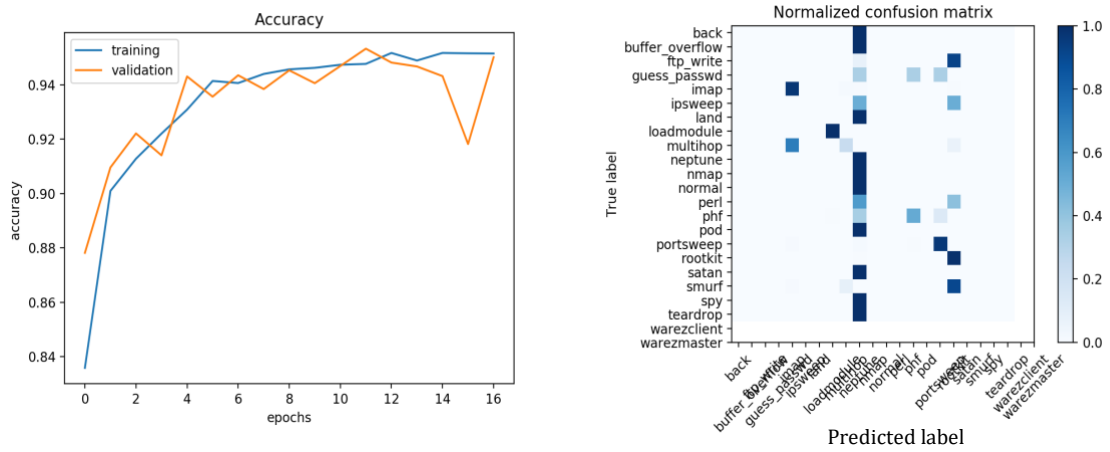

Figure 11. NSL-KDD approach 0 accuracy graphic and confusion matrix

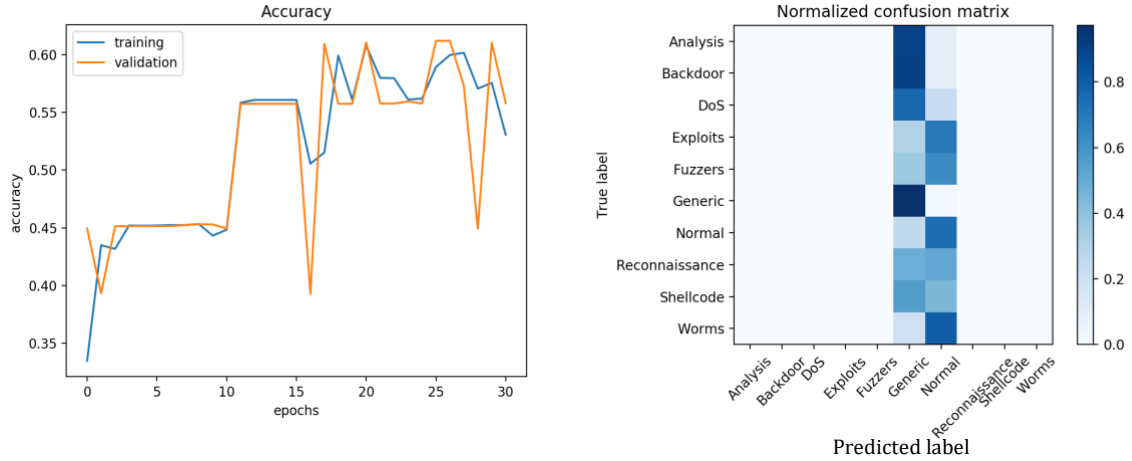

Figure 12. UNSW-NB15 approach 0 accuracy graphic and confusion matrix

Observing Figure 9, it is possible to see that the network is not working correctly for the UGR16 when data is not preprocessed and so the validation score is that low. It can also be noted that the number of epochs is another indicator of the failure of the test.

According to Figure 8, UGR16 *approach 0_op2* has the best accuracy when nothing is preprocessed and moreover for this concrete approach, source and destination IP are removed. It can be observed in the confusion matrix in Figure 10 that just anomaly-spam and background are detected.

Confusion matrix in Figure 11 for NSL-KDD shows a weak diagonal agreeing with the not so unsatisfactory accuracy value.

There is not much information given by Figure 12. As it is possible to observe in both graphics for UNSW-NB15, accuracy is relatively low and only generic and normal traffic had been approximately detected.

The detailed results can be consulted in Appendix B.

The procedures carried out next are exposed bellow.

4.5.1. FIRST APPROACH

Application of the same method to all the features.

UGR16

- When applying zscore normalization (*zscore_all_1)* the validation score obtained is 0,99332.
- When applying minmax between 0 and 1 (*minmax_0_all_1)* the validation score obtained is 0,98880.
- When applying minmax between -1 and 1 (*minmax_1_all_1)* the validation score obtained a 0,99185.

NSL-KDD

- When applying zscore normalization (*zscore_all_2)* the validation score obtained a 0,97898.
- When applying minmax between 0 and 1 (*minmax_0_all_2)* the validation score obtained a 0,96253.
- When applying minmax between -1 and 1(*minmax_1_all_2)* the validation score obtained a 0,96383.

UNSW-NB15

- When applying zscore normalization (*zscore_all_3)* the validation score obtained a 0,81042.
- When applying minmax between 0 and 1 (*minmax_0_all_3)* the validation score obtained a 0,75669.
- When applying minmax between -1 and 1 (*minmax_1_all_3)* the validation score obtained a 0,79944.

It is shown below the accuracy graphic and the normalized confusion matrix, for the *zscore_all_1, zscore_all_2* and *zscore_all_3* tests because they have the higher results. A summarizing graphic for the first approach over the three datasets is also shown.

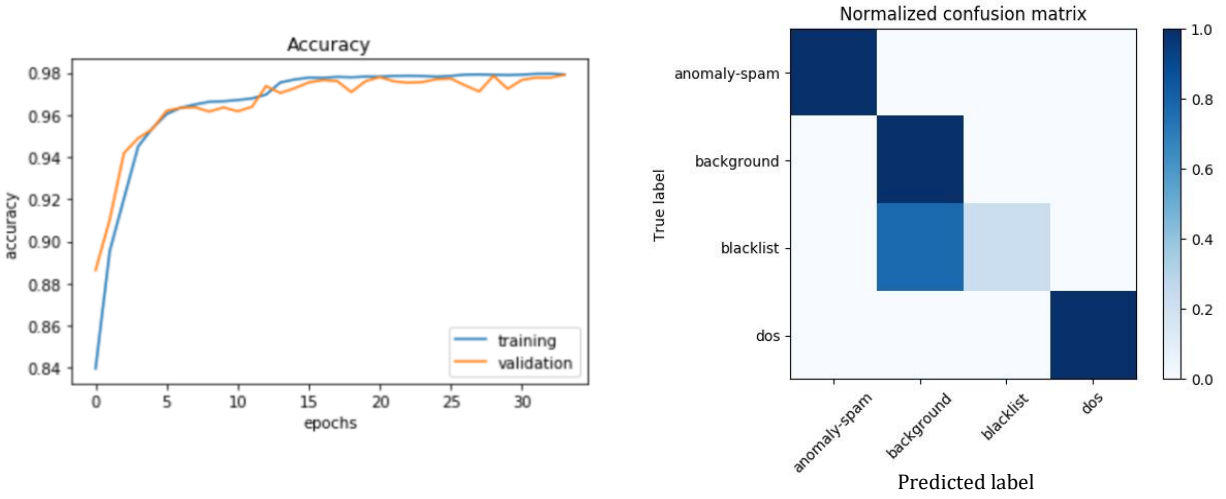

Figure 13. UGR16 zscore_all_1 accuracy and confusion matrix

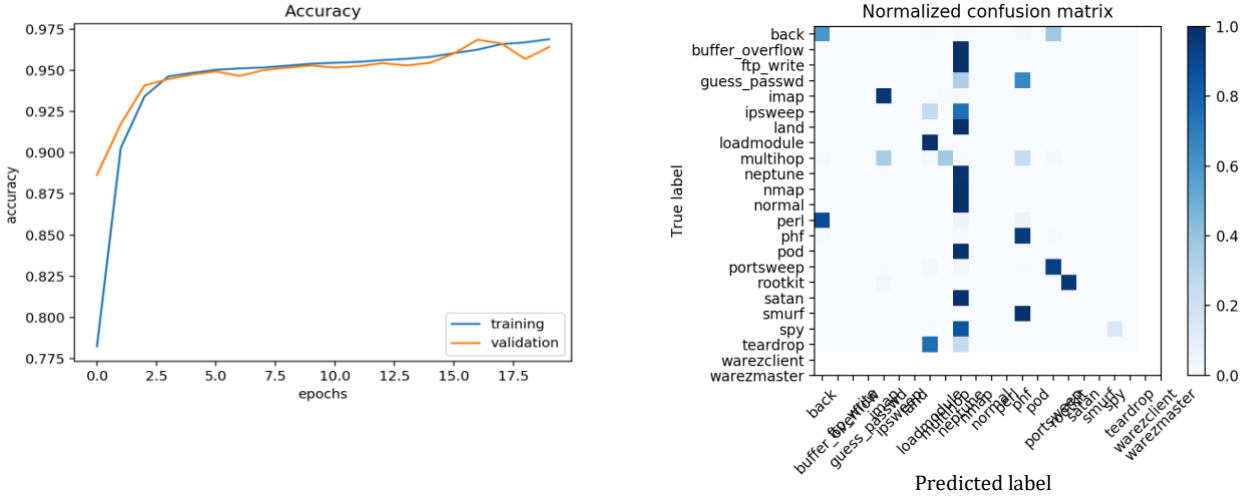

Figure 14. NSL-KDD zscore_all_1 accuracy and confusion matrix

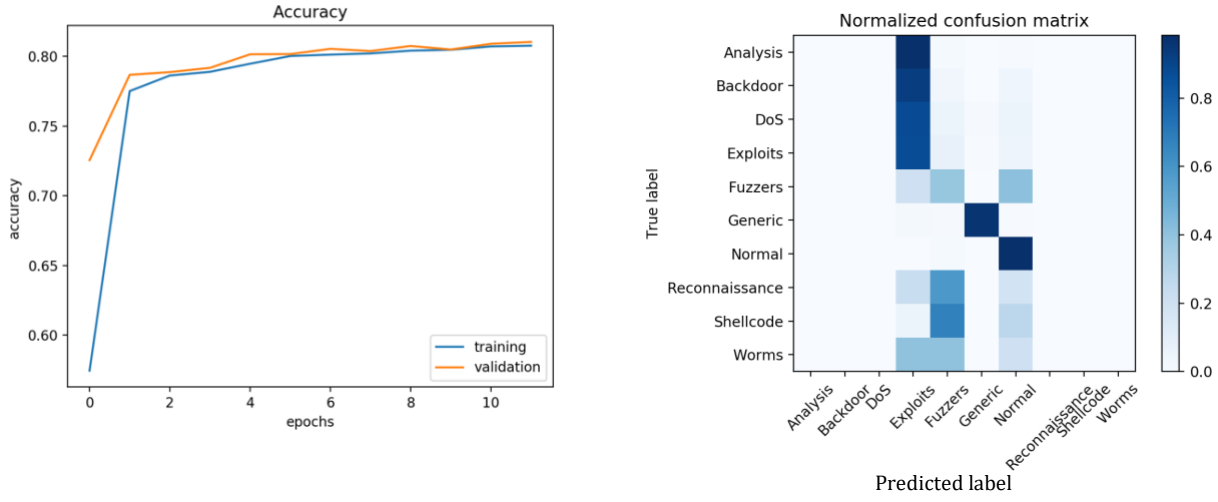

Figure 15. UNSW-NB15 zscore_all_3 accuracy and confusion matrix

For the three cases exposed above, as expected, it is possible to observe that for both training and validation sets, the accuracy increases with each epoch. On the other hand, observing the confusion matrixes obtained, it can be seen a clear diagonal for the UGR16, taking into account that some of the

blacklist labels have been identified as background. For the NSL-KDD it is possible to observe a week diagonal, having most of the labels identified as normal traffic. Last, the UNSW-NB15 has made worse predictions, it is not possible to identify any diagonal and it can be appreciated that the best predictions have been with generic and normal labels.

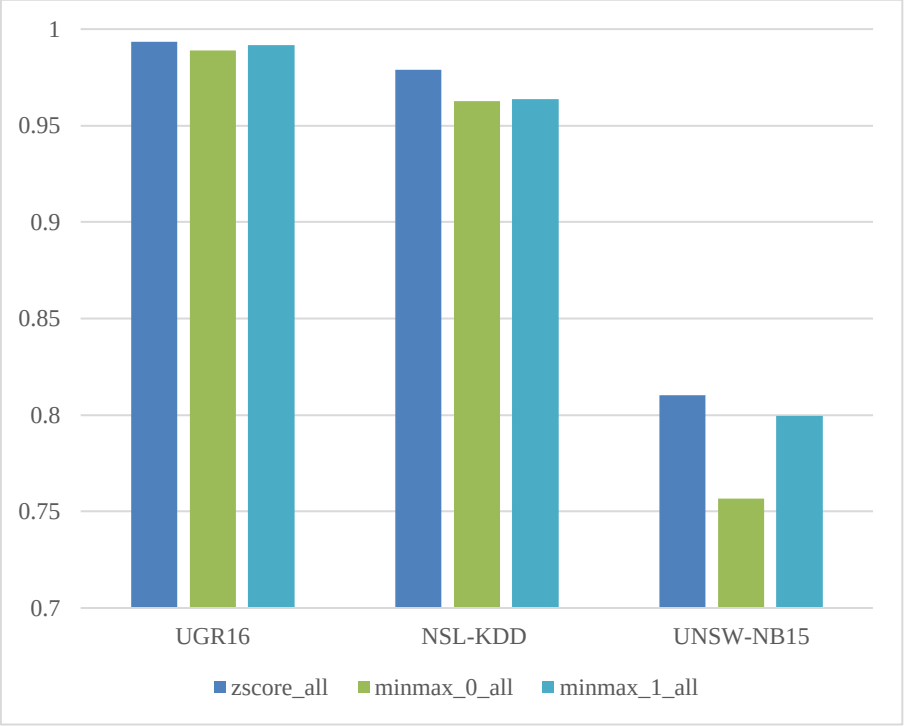

Figure 16. Validation scores for approach 1

As it can be observed, ***zscore_all* gets better results in the three datasets**, followed by *minmax_1_all* and then *minmax_0_all*.

All the detailed validation scores for *approach 1* can be consulted in Appendix B.

4.5.2. SECOND APPROACH

For this procedure two different points of view have been considered. As it was already mentioned, the second approach can be categorized in two types, depending on the dataset evaluated. Different options have been tested on NSL-KDD and UNSW-NB15, variating the method applied on the different grouped features. However, UGR16 has been diversely preprocessed in function of the individual features.

UGR16 second approach has been the simplest. Three variations have been made, *Var1_1*, *Var2_1* and *Var3_1*, in which *sip* and *dip* features has been encoded with the three normalization options available. The results obtained are very similar for *Var1_1*and *Var2_1* and *Var_3*, in which labels are **normalized with zscore** function, has the highest accuracy.

The accuracy graphic and confusion matrix for *Var_3* are shown below.

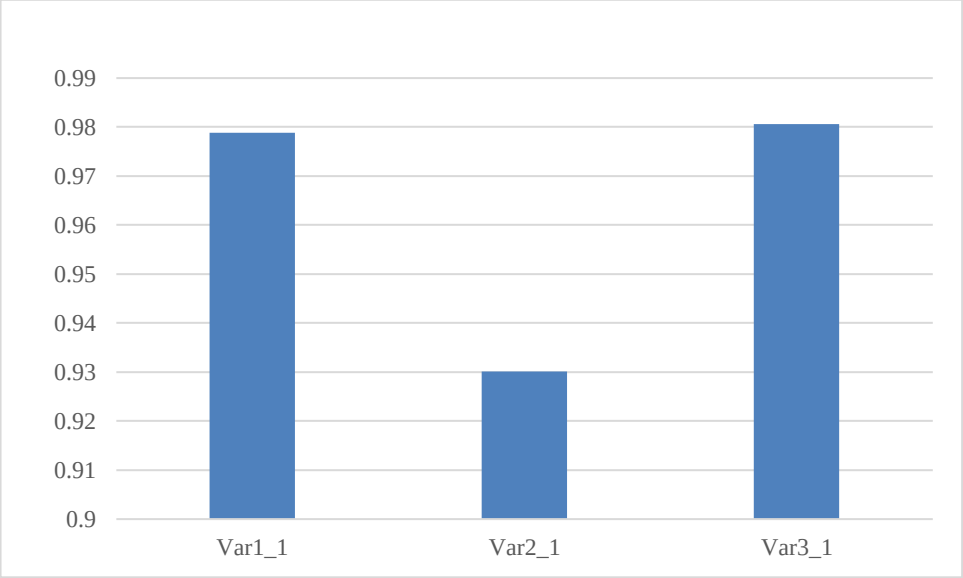

Figure 17. UGR16 validation score for approach 2

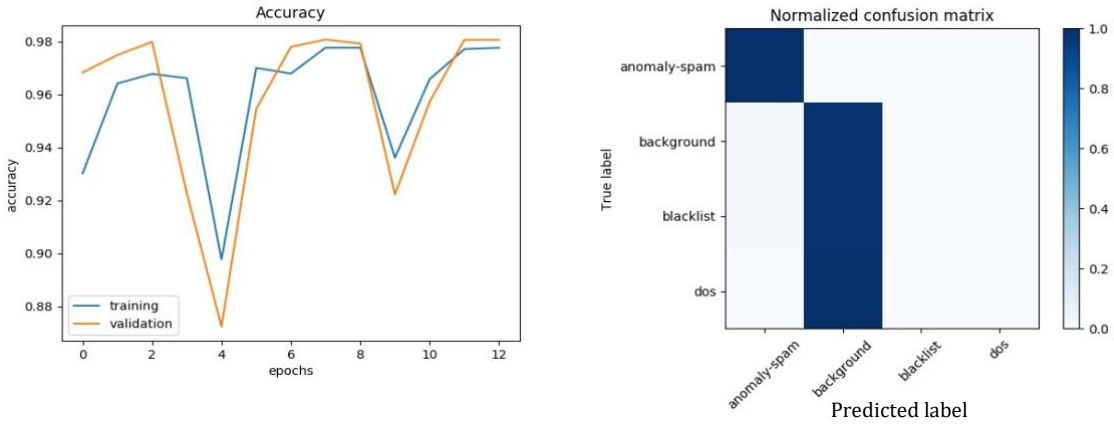

Figure 18. UGR16 Var_3 accuracy and confusion matrix

The confusion matrix shows that all of the anomaly-spam and background labels have been detected correctly although the other labels have been identified as background traffic. The evolution of the accuracy and its highest and lowest points can be seen in the graphic too.

For the NSL-KDD dataset, nine options have been developed. In the following table the results obtained are shown, sorted by validation score.

Table 14. Variations over NSL-KDD approach 2 in descendant order

Nº variation	Basic features	Content features	Traffic features	Validation score
Var10_2	zscore	-	zscore	0,97923414
Var5_2	min_max_0	min_max_0	zscore	0,978567346
Var6_2	zscore	min_max_0	min_max_0	0,97694799
Var1_2	min_max_0	zscore	zscore	0,974693592
Var11_2	zscore	zscore	-	0,971550137
Var4_2	min_max_0	zscore	min_max_0	0,970851591
Var9_2	-	-	zscore	0,970121293
Var12_	-	zscore	zscore	0,969867276
Var3_2	zscore	zscore	min_max_0	0,964628183
Var8_2	-	zscore	-	0,946434241
Var7_2	zscore	-	-	0,935924303
Var2_2	zscore	min_max_0	zscore	0,007048962

First fact to be appreciated has been that the best results are obtained for the variations in which all the groups of features, or most of them, are encoded. Specifically, the Basic features group. It can be also noticed that Var2_2 is not working and the accuracy is near 0.

Confusion matrixes for the best and the worst variations are shown below, considering Var7_2 the worst one.

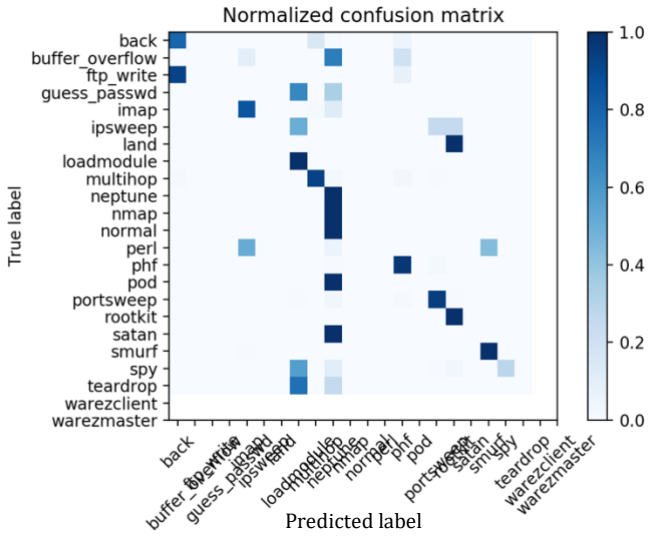

Figure 19. Var10_2 confusion matrix

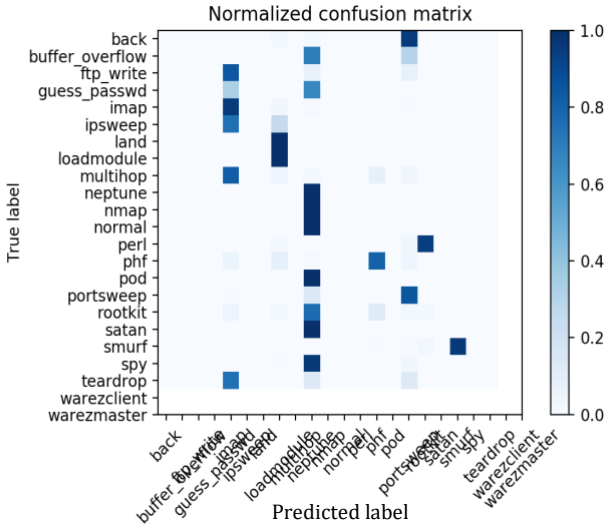

Figure 20. Var7_2 confusion matrix

According to the validation scores for these variations, it is easier to appreciate the diagonal in Figure 13 although Figure 14 is pretty easy to see it too due to the fact that the value obtained is also high.

The second approach with the UNSW-NB15 has been made with 9 variations. The table below shows the results sorted by validation score and the method applied to each group in each variation.

Table 15. Variations over UNSW-NB15 approach 2 in descendant order

Nº variation	Basic features	Time features	Content features	Add. Features	Validation scores
Var4_3	zscore	zscore	zscore	-	0,822086188
Var1_3	zscore	min_max_0	min_max_0	min_max_0	0,820871593
Var2_3	zscore	-	-	-	0,71384152
Var6_3	-	-	zscore	-	0,711849585
Var3_3	zscore	zscore	-	-	0,684982753
Var9_3	-	zscore	zscore	zscore	0,606908614
Var8_3	-	zscore	zscore	-	0,576397998
Var5_3	-	-	-	zscore	0,55817908
Var7_3	-	zscore	-	-	0,557596074

From Table 15 it can be inferred that encoding the *Basic features* not only gives the best validation scores but not encoding them makes the score go down considerably. Following the same procedure as with NSL-KDD, the confusion matrixes for the best test, *Var4_3*, and the worst one, *Var7_3*, are shown in the following figures.

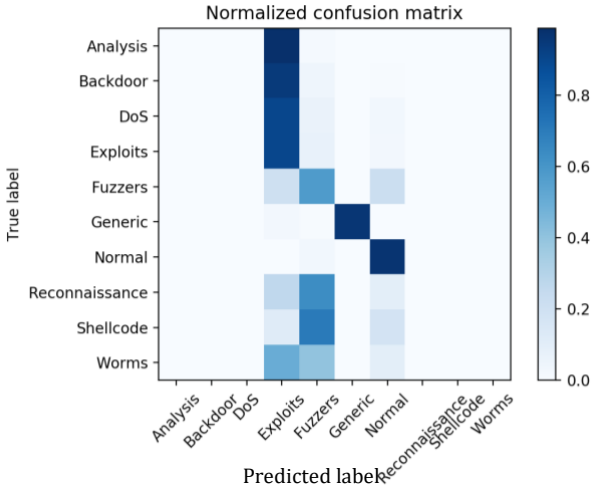

Figure 21. Var4_3 confusion matrix

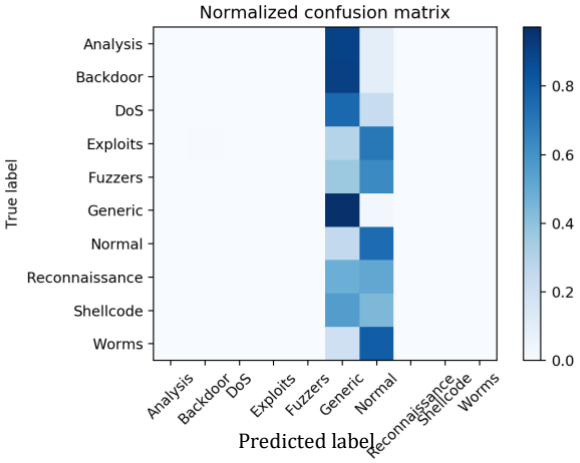

Figure 22. Var7_3 confusion matrix

For *Var4_3*, having around an 80% of accuracy, the labels the best predicted have been Exploits, Fuzzers, Generic and Normal whereas for *Var7_3*, the accuracy has been lower than in the *approach 0.* As it can be seen, no information is given by Figure 22.

Summarizing figures are shown below with the results of all the approaches made compared to the percentage of normal traffic for each dataset. Contrasting this percentage to the predictions made is another valid evaluation method in such a way that when the validation score is higher than the percentage it involves predictions have been done, considering later how accurate has been that prediction.

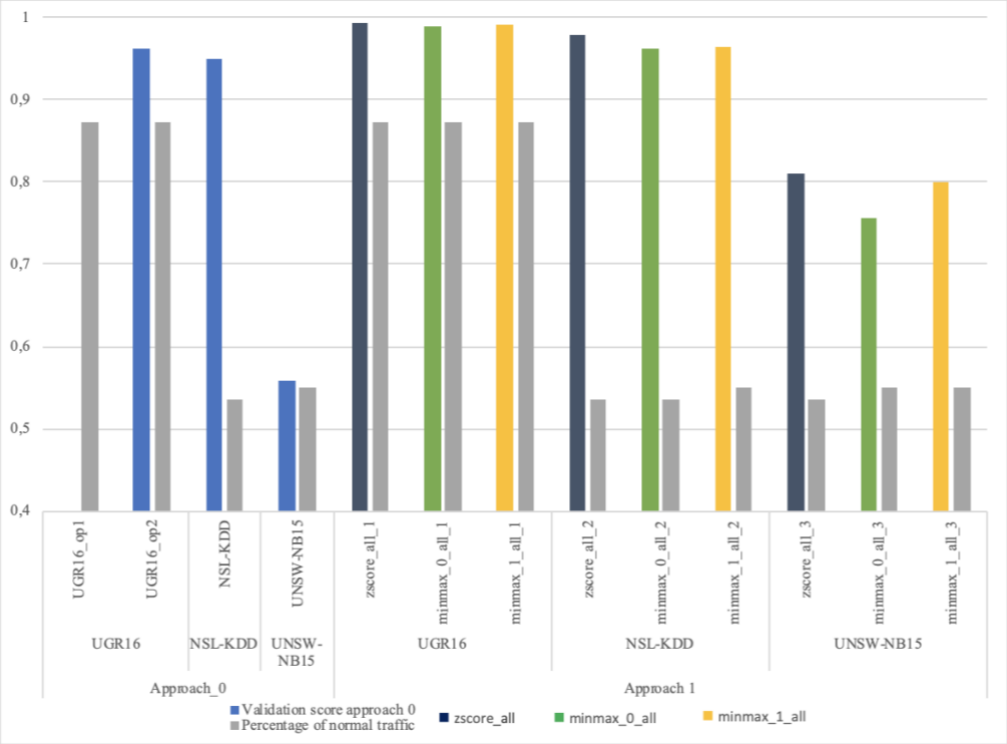

Figure 23. Approach 0 and approach 1 results

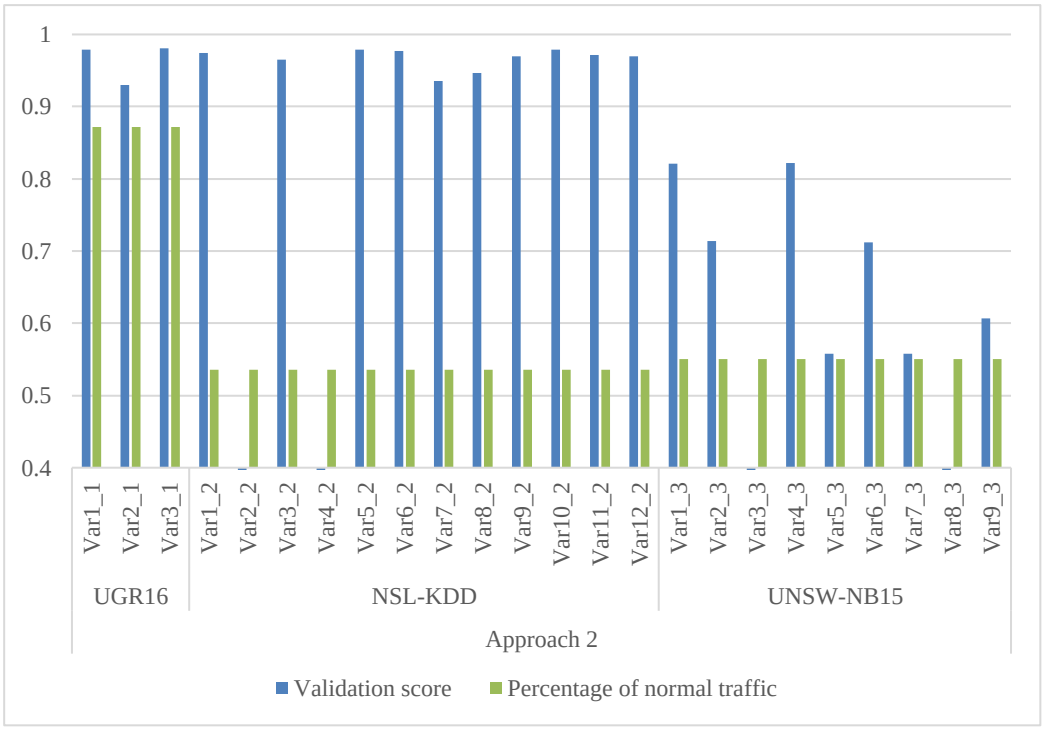

Figure 24. Approach 2 results

5. CONCLUSIONS AND FUTURE WORKS

Conclusions extracted and future work lines are described in below.

5.1. CONCLUSIONS

In the following section the conclusions achieved are going to be formulated even though some of them have been exposed in this case study.

First conclusion, which applies to all the datasets studied, is that the transformation into numerical values is a requirement, not just for this project's aim, but for the model to be able to run, having the possibility to encode the features to *dummy variables* or to *indexes*. Once the data is numerical, it is possible to normalize it with different techniques, being *zscore* and *minmax* the ones chosen for this study.

Second conclusion, which also involves the three datasets, is that the best results are granted when all the features are preprocessed. If the same normalization method is used, the best results are obtained for *zscore*. The reason for this is that most of the features in the different datasets have distanced values (outliers) e.g., IP addresses or ports, and *zscore* usually offers better results when outliers are common. The second-best results are obtained when using *minmax between -1 and 1*.

The conclusion formulated for the UGR16 dataset, added to the conclusions previously exposed, is that the results does not differ too much between *zscore* and *minmax between -1 and 1*, 97% and 98% respectively, as it can be verified in Figure 17- *UGR16 validation scores for approach 2*, and they are very accurate.

The fourth conclusion applies only to NSL-KDD and UNWS-NB15, and is related to the groups of features in each dataset, being these groups classified as *Basic features, Content features* and *Traffic features* for the first one, and *Basic features, Time features, Content features* and *Additional generated features* for the second one. As shown in tables 14 and 15, in which variations by groups over NSL-KDD and UNSW-NB15 are listed, the best validation scores in general are obtained when more groups are encoded with *zscore*, being around 97% for NSL-KDD and 82% for UNSW-NB15. However, the higher results among all variations are when *basic features* are encoded. It is possible to notice in tables 5 and 7, which contain the features included in each group, that NSL-KDD and UNSW-NB15 common *basic features* are duration, protocol, service, source to destination bytes and destination to source bytes. The fact by which these results are obtained is because *basic features* offer more relevant information than the other groups thus preprocessing them makes the model more accurate.

For NSL-KDD is concluded, in addition to the paragraph above, that normalizing *content features* do not make much difference through the variations performed because they include concrete information for determined attacks. Furthermore, encoding *traffic features* is relevant as it is the group with more features, so it gives abundant information to the network.

Last conclusion involves UNSW-NB15. It is also concluded for this dataset that preprocessing *time features* and *additional generated features* do not contribute to the improvement of the validation score, in fact, the worst results are for the variations with only these groups encoded. This can be explained due to, on the one hand, *time features* are just 7 among 45, and on the other, *additional generated features* are, as its name indicates, additional data that does not give relevant information.

The main purpose of this project has been accomplished for the UGR16, NSL-KDD and UNSW-NB15, the chosen datasets. Its features have been analyzed, preprocessing methods, such as *zscore* and *minmax normalization*, have been applied and the different methods proposed have been evaluated so it has been possible to propose a data preprocessing model for each of them.

The code used in this project is available on GitHub.

Last, major problems encountered are going to be related.

The general issue faced has been acquiring enough machine learning knowledge to develop this project.

The main obstacle has been treating with the UGR16 dataset because of its 81GB original size. It has been necessary to use high computing capabilities for processing it and so Google Cloud Engine has been employed. However, even with this facility, it has not been possible to use the original one as more machine learning knowledge is needed, not being enough with the programming language abilities. In consequence, a portion of the UGR16 has been used as it has been exposed in the previous sections.

Another problem confronted is related with UNSW-NB15 and the fact that its output feature is binary whereas this project is focused on multiclass problems.

5.2. FUTURE WORKS

In order to improve the project presented in this document, various possible works are proposed in this section.

- **Preprocessing analysis with other machine learning techniques.** Based on the fact that preprocessing with *zscore* all the features offers the accurate results, a preprocessing analysis could be done with diverse neural networks with different characteristics.

- **Utilization of the entire UGR16 dataset.** As this project is not focused on dealing with huge quantities of data, the preprocessing analysis using the original UGR16 and the comparison of the results is an interesting option.

- **Analysis of the preprocessing model dealing with missing features.** Studying the relevance of the different missing features techniques: removing data or supplying it.

- **Comparison between UNSW-NB15 approaches as a multiclassification problem and as a binary problem.** Since this project has used UNSW-NB15 simulating a multiclassification problem, a comparison of results between multiclass and binary can be done.

6. BIBLIOGRAPHY

[1] Paul Innella, Tetrad Digital Integrity, LLC, «The Evolution of Intrusion Detection Systems,» pp. 1-4, 2001.

[2] H. Kaur, G. Singh y J. Minhas, «A Review of Machine Learning based Anomaly Detection Techniques,» *International Journal of Computer Applications Technology and Research,* 2013.

[3] S. Russell y P. Norvig, «Artificial Intelligence. A modern approach,» Pearson.

[4] Cyber Security, «MIT Lincoln Laboratory,» 1998. [En línea]. Available: https://www.ll.mit.edu/r-d/datasets.

[5] G. Maciá-Fernández, J. Camacho, R. Magán-Carrión, P. García-Teodoro y R. Therón, «UGR'16: A new dataset for the evaluation of cyclostationarity-based network IDSs,» *ScienceDirect,* 2017.

[6] M. Tavallaee, E. Bagheri, W. Lu y A. A. Ghorbani, «A Detailed Analysis of the KDD CUP 99 Data Set,» de *Proceedings of the 2009 IEEE Symposium on Computational Intelligence in Security and Defense Applications (CISDA 2009)*, 2009.

[7] G. Creech y J. Hu, de *IEEE Wireless Communications and Networking Conference (WCNC)*, 2013.

[8] G. S, G. M, S. J y Z. A, «An empirical comparison of botnet detection method,» 2014.

[9] N. Moustafa y J. Slay, «UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set),» 2015.

[10] J. Huang, Y.-F. Li y M. Xie, «An empirical analysis of data preprocessing for machine learning-based software cost estimation,» pp. 1-4, 2015.

[11] C. N. Modi y P. D. Patel, «A novel Hybrid-Network Intrusion Detection Systems (H-NIDS) in Cloud Computing,» p. 5.

[12] Y. Li, B. Fang, L. Guo y Y. Chen, «Network Anomaly Detection Based on TCM-KNN Algorithm,» 2007.

[13] W. Ma, D. Tran y D. Sharma, «A Study on the Feature Selection of Network Traffic for Intrusion Detection Purpose,» 2008.

[14] W. Wang, X. Zhang, S. Gombault y S. J. Knapskog, «Attribute Normalization in Network Intrusion Detection,» de *10th International Symposium on Pervasive Systems, Algorithms, and Networks*, 2009.

[15] S. kumarl y A. Yadav2, «Increasing Performance Of Intrusion Detection System Using Neural Network,» de *IEEE International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*, 2014.

[16] N. Lokeswari y B. C. Rao, «Artificial Neural Network Classifier for Intrusion Detection System in Computer Network,» de *Second International Conference on Computer and Communication Technologies, Advances in Intelligent Systems and Computing*, 2016.

[17] Z. Chiba, N. Abghour, K. Moussaid, A. E. Omri y M. Rida, «A novel architecture combined with optimal parameters for back propagation neural networks applied to anomaly network intrusion detection,» *ScienceDirect,* 2018.

[18] A. R. Deepika P Vinchurkar, «A Review of Intrusion Detection System Using Neural Network and Machine Learning Technique,» *International Journal of Engineering Science and Innovative Technology (IJESIT)*, pp. 56-58, 2014.

[19] T. M. Michell, *The Discipline of Machine Learning*, Machine Learning Department School of Computer Science Carnegie Mellon University Pittsburgh, PA 15213, 2006.

[20] H. Debar, M. Becker y D. Simone, *A Neural Network Component for Intrusion Detection Systems.*

[21] S. Agatonovic-Kustrin y R. Beresford, «Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research,» *ScienceDirect*, 2000.

[22] G. Kumar, K. Kumar y M. Sachdeva, *The use of artificial intelligence based techniques for intrusion detection: a review*, 2010.

[23] V. R, S. KP y P. Poornachandran, *Applying Convolutional Neural Network for Network Intrusion Detection.*

[24] S. B. Kotsiantis, D. Kanellopoulos y P. E. Pintelas, «Data Preprocessing for Supervised Leaning,» *INTERNATIONAL JOURNAL OF COMPUTER SCIENCE*, 2006.

[25] A. Burkov, «Chapter 5,» de *The Hundred-Page Machine Learning Book Draft*.

[26] J. Brownle, «How to Stop Training Deep Neural Networks At the Right Time Using Early Stopping,» 2018. [Online]. Available: https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/.

[27] «NumPy,» [Online]. Available: https://www.numpy.org/.

[28] «SciPy,» [Online]. Available: https://www.scipy.org/.

[29] «Matplotlib,» [Online]. Available: https://www.matplotlib.org/.

[30] «Scikit-learn,» [Online a]. Available: https://scikit-learn.org/stable/.

[31] «Pandas,» [Online]. Available: https://pandas.pydata.org/.

[32] «Conda,» [Online]. Available: https://docs.conda.io/en/latest/.

[33] «Jupyter Notebook,» [Online]. Available: https://jupyter.org/.

[34] «Google Compute Engine,» [Online]. Available: https://cloud.google.com/compute/?hl=es.

[35] J. Heaton, «Jeff Heaton Deep Learning,» [Online]. Available: https://github.com/jeffheaton/t81_558_deep_learning.

[36] «pandas.DataFrame.drop,» [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.drop.html.

[37] «pandas.get_dummies,» [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html.

APPENDIX A: FEATURES DESCRIPTION OF EACH DATASET

Table 16. UGR16 features description

No.	Name	Dtype	Description
1	time	object	Timestamp of the en of a flow
2	duration	float64	Duaration of the flow
3	sip	object	Source IP address
4	dip	object	Destination IP address
5	source port	int64	Source port
x6	dest port	int64	Destination port
7	protocol	object	Protocol
8	flags	object	Flags
9	forward status	int64	Forwarding status
10	type service	int64	Type of service
11	pack_exanged	int64	Packets exchanged in the flow
12	bytes	int64	Their corresponding number of bytes
13	attack_tag	object	Type of attack

Table 17. NSL-KDD features description

No.	Name	dtype	Category	Description
1	duration	real	Basic features	Duration of the connection (seconds)
2	protocol_type	texto	Basic features	Type of protocol (TCP, UPD, ICMP, etc.)
3	service	texto	Basic features	Network Service (http, telnet, https, others)
4	flag	texto	Basic features	Connection status (SF, S0, S1, S2, S3, OTH, REJ, RSTO, RSTO,S0, SH, RSTRH, SHR)
5	src_bytes	real	Basic features	Number of bytes sent from source to destination
6	dst_bytes	real	Basic features	Number of bytes received from destination
7	land	bin	Basic features	If source and destination IP are identical. It is 1 else 0
8	wrong_fragment	real	Basic features	Sum of Bad checksum packets in a connection
9	urgent	real	Basic features	Some of packets where urgent bit is set 1.
10	hot	real	Content features	Sum of hot actions in a connection (entering a system directory, creating programs and executing programs)
11	num_failed_logins	real	Content features	Number of failed logins in a connection
12	logged_in	bin	Content features	1 if successful login else 0.

13	num_compromised	real	Content features	Sum of not found error appearances in a connection
14	root_shell	real	Content features	1 if root shell is obtained else 0
15	su_attempted	real	Content features	Its 1 if su command attempted else 0
16	num_root	real	Content features	Sum of operations performs as a root in a connection
17	num_file_creations	real	Content features	Sum of file creations in a connection
18	num_shells	real	Content features	Number of shell prompts
19	num_access_files	real	Content features	Sum of operations in control files in a connection
20	num_outbound_cmds	real	Content features	Sum of outbound commands in a ftp session
21	is_host_login	bin	Content features	If the user is accessing s root , it is set to 1 else 0
22	is_guest_login	bin	Content features	If the user is accessing s guest, it is set to 1 else 0
23	count	real	Traffic features	Sum of connections to the same destination IP
24	srv_count	real	Traffic features	Sum of connections to the same destination Port no.
25	serror_rate	real	Traffic features	Percentage of connections that have activated the flag (P4) s0, s1, s2 or s3, among the connections aggregated in count (P23)
26	srv_serror_rate	real	Traffic features	Percentage of connections that have activated the flag (P4) s0,
27	rerror_rate	real	Traffic features	Percentage of connections that have activated the flag (P4) REJ, among the connections aggregated in count (P23)
28	srv_rerror_rate	real	Traffic features	Percentage of connections that have activated the flag (P4) REJ, among the connections aggregated in count (P23)
29	same_srv_rate	real	Traffic features	Percentage of connections that were to the same service, among the connections aggregated in count (P23)
30	diff_srv_rate	real	Traffic features	Percentage of connections that were to different services, among the connections aggregated in count (P23)
31	srv_diff_host_rate	real	Traffic features	Percentage of connections that were to different destination machines among the connections aggregated in srv count (P24)
32	dst_host_count	real	Traffic features	Sum of connections to the same destination IP address
33	dst_host_srv_count	real	Traffic features	Sum of connections to the same destination port number
34	dst_host_same_srv_rate	real	Traffic features	Percentage of connections that were to the same service, among the connections aggregated in dst host count (P32)
35	dst_host_diff_srv_rate	real	Traffic features	Percentage of connections that were to different services, among the connections aggregated in dst host count (P32)
36	dst_host_same_src_port_rate	real	Traffic features	Percentage of connections that were to the same source port, among the connections aggregated in dst host srv count (P33)

37	dst_host_srv_diff_host_rate	real	Traffic features	Percentage of connections that were to different destination machines, among the connections aggregated in dst host srv count (P33)
38	dst_host_serror_rat	real	Traffic features	Percentage of connections that have activated the flag (f4) s0, s1, s2 or s3, among the connections aggregated in dst host count (P32)
39	dst_host_srv_serror_rate	real	Traffic features	Percent of connections that have activated the flag (P4) s0, s1, s2 or s3, among the connections aggregated in dst host srv count (P33) or s3, among the connections aggregated in dst host srv count (P33)
40	dst_host_rerror_rate	real	Traffic features	Percentage of connections that have activated the flag (P4) REJ, among the connections aggregated in dst host count (P32)
41	dst_host_srv_rerror_rate	real	Traffic features	Percentage of connections that have activated the flag (P4) REJ, among the connections aggregated in dst host srv count (P33)
42	attack_tag	texto		The name of each attack category.

Table 18. UNSW-NB features description

No.	Name	dtype	Group	Description
1	id	int64		ID
2	dur	float64	Basic features	Row total duration.
3	proto	object	Flow features	Protocol type, such as TCP, UDP.
4	service	object	Basic features	Such as http, ftp, smtp, ssh, dns and ftp-data.
5	state	object	Basic features	The states and its dependent protocol e.g., CON
6	spkts	int64	Basic features	Source to destination packet coun
7	dpkts	int64	Basic features	Destination to source packet count.
8	sbytes	int64	Basic features	Source to destination bytes.
9	dbytes	int64	Basic features	Destination to source bytes
10	rate	float64		The content size of the data transferred from http.
11	sttl	int64	Basic features	Source to destination time to live.
12	dttl	int64	Basic features	Destination to source time to live.
13	sload	float64	Basic features	Source bits per second.
14	dload	float64	Basic features	Destination bits per second.
15	sloss	int64	Basic features	Source packets retransmitted or dropped
16	dloss	int64	Basic features	Destination packets retransmitted or dropped.
17	sintpkt	float64	Time features	Source inter-packet arrival time.
18	dintpkt	float64	Time features	Destination inter-packet arrival time.

19	djit	float64	Time features	Source jitter.
20	sjit	float64	Time features	Destination jitter.
21	swin	int64	Content features	Source TCP window advertisement value
22	stcpb	int64	Content features	Source TCP base sequence number.
23	dtcpb	int64	Content features	Destination TCP base sequence number
24	dwin	int64	Content features	Destination TCP window advertisement value.
25	tcprtt	float64	Time features	Setup round-trip time, the sum of 'synack' and 'ackdat'.
26	synack	float64	Time features	The time between the SYN and the SYN_ACK packets
27	ackdat	float64	Time features	The time between the SYN_ACK and the ACK packets
28	smean	int64	Content features	Mean of the packet size transmitted by the srcip.
29	dmean	int64	Content features	Mean of the packet size transmitted by the dstip.
30	trans_depth	float64	Content features	The connection of http request/response transaction.
31	response	float64	Content features	The content size of the data transferred from http.
32	ct_srv_src	float64	Additional generated features	No. of rows of the same service (14) and srcip (1) in 100 rows.
33	ct_state_ttl	int64	Additional generated features	No. of each state (6) according to values of sttl (10) and dttl (11).
34	ct_dst_ltm	int64	Additional generated features	No. of rows of the same dstip (3) in 100 rows.
35	ct_src_dport_ltm	int64	Additional generated features	No of rows of the same srcip (1) and the dsport (4) in 100 rows.
36	ct_src_sport_ltm	int64	Additional generated features	No of rows of the same dstip (3) and the sport (2) in 100 row
37	ct_dst_src_ltm	object	Additional generated features	No of rows of the same srcip (1) and the dstip (3) in 100 rec
38	is_ftp_login	int64	Additional generated features	If the ftp session is accessed by user and password then 1 else 0.
39	ct_ftp_cmd	int64	Additional generated features	No of flows that has a command in ftp session.
40	ct_flw_http_mthd	int64	Additional generated features	No. of methods such as Get and Post in http service
41	ct_src_ltm	int64	Additional generated features	No. of rows of the srcip (1) in 100 rows.
42	ct_srv_dst	int64	Additional generated features	No. of rows of the same service (14) and dstip (3) in 100 row
43	is_sm_ips_ports	int64	Additional generated features	If srcip (1) = dstip (3) and sport (2) = dsport (4), assign 1 else 0.
44	attack_cat	object	Labelled features	The name of each attack category.
45	Label	int64	Labelled features	0 for normal and 1 for attack records

APPENDIX B: APPROACH 1 AND 2 RESULTS

Table 19. Approach 0 validation scores

Dataset	Validation scores for approach 0
UGR16_op1	0,0070011
UGR16_op2	0,961395902
NSL-KDD	0,950180987
UNSW-NB15	0,558033328

Table 20. Approach 1 validation scores

Dataset	zscore_all	minmax_0_all	minmax_1_all
UGR16	0,99332981	0,988801058	0,991857237
NSL-KDD	0,97898012	0,962532546	0,963834381
UNSW-NB15	0,81042608	0,756692416	0,799446145

Table 21. Approach 2 for UGR16 validation scores

UGR16	Validation score
Var1_1	0,978842036
Var2_1	0,930141441
Var3_1	0,980581626

APPENDIX C: ETHICAL ECONOMIC, SOCIAL AND ENVIRONMENTAL ASPECTS

In this appendix, the general impact of this project is going to be detailed, considering ethical, economic, social and environmental aspects.

C.1 INTRODUCTION

As it has been explained along the present document, this project involves Intrusion Detection Systems and Machine Learning, proposing an efficient data preprocessing model of IDS based on Deep Learning with the goal of reducing the false positives rate and increase the accuracy of them.

C.2 DESCRIPTION OF MORE RELEVANT IMPACTS IN RELATION WITH THE PROJECT

Intrusion detection is a fundamental task in cybersecurity, that nowadays has an important impact in the society and the economy, especially of the companies. People are calmer if they know that systems that detects intrusions to their systems are the efficient as possible and companies can reduce their outgoings. The last idea mentioned is going to be explained bellow.

As most of involving Machine Learning projects, this one has a relevant ethical impact in relation with the fact that the purpose of this discipline is to make machines capable of taking decisions. This will be detailed in the following paragraph.

In relation with the environmental impact, it has been considered the system proposed is supposed to be working 24h 7 days per week due to its aim of detecting intrusions therefore there is a need of high electric power supply.

C.3 DETAILED ANALYSIS OF SOME OF THE MAIN IMPACTS

Economic and ethical are the two main impacts considered.

Economic impact, on the one hand, is related to the fact that this model proposed presents better results than existing models or systems with the same purpose so it will have a positive economic impact. On the other hand, the repercussions in a company will be also positive in terms of reducing resources and time to monitor systems and networks.

The previous mentioned idea is related with people feeling safe knowing that the methods used to prevent attacks are the accurate they can.

Ethical impact has been also considered as important since Machine learning has the goal of make machines learn and make perditions without human intervention, jobs usually carried out by persons will be executed for machines.

C.4 CONCLUSIONS

To conclude, this project will have a majorly positive impact considering as principal economic and social impact.

APPENDIX D: ECONOMIC BUDGET

This appendix details an economic budget for the development of this project, including the physical resources, the human resources, licenses and taxes.

D.1 PHYSICAL RESOURCES

For developing an important part of this project, it is required a computer at least with the following characteristics and an approximated cost of 1300€:

- **CPU**: 2,3GHz Intel Core i5
- **RAM:** 8GB
- **DISK:** 256GB SDD

Estimating that the lifespan of a computer is 3 years and that the duration of the project is 4 months, only 16% of the computer will be used, being the real cost **208€**.

Moreover, it is necessary to use Google Cloud Engine to create an environment with the following characteristics:

- **CPU:** automatic generated 8vCPUs
- **RAM:** 30GB
- **DISCK:** 1TB

The cost of this platform is estimated in **300€.**

D.2 HUMAN RESOURCES

For the estimation of the human resources costs, it is considered that the project is carried out by a single person. According to the limits established by the Universidad Politécnica de Madrid, 25h worked per week is the maximum for an undergraduate student. If the average salary of an internship with this schedule is 500€ per month, the costs of human resources will amount **2000€**.

D.3 LICENSES

The software used to develop this project is open-source based and free so there will not be license costs.

D.4 TAXES

In case that the project is sold to another company, the taxes derived of the transactions are a 15% over the final cost.

Whit all these considerations, the total cost of the project amounts to **2884,2€** and it is summarized in the following table:

Physical resources	508€
Human resources	2000€
Licenses	0
Taxes	376,2
Total	**2884,2**