Politechnic University of Bari

Department of Electrical and Information Engineering

Master’s degree in Computer Science Engineering

Thesis in

Advanced Software Engineering

Cyber-Attack Mitigation in Cloud-Fog Environment: a comprehensive overview, research directions and proposed approaches

Supervisor:

Prof. Marina **MONGIELLO**

Co-Supervisor:

Prof. Francesco **NOCERA**

Candidate:

Dott. Pietro **URSO**

Academic Year 2021 – 2022

Abstract in Italian

Nessuno è al sicuro sul web, gli attacchi informatici sono sempre più frequenti e le aziende, soprattutto quelle di grandi dimensioni, sono sotto l'occhio dei cyber criminali.

Inoltre, le tecnologie cloud si sono diffuse in modo esponenziale nel mondo, per cui sono emerse grandi preoccupazioni riguardo alla sicurezza informatica e alla privacy delle aziende e anche delle singole persone che utilizzano i servizi dei provider cloud. Il numero di attacchi informatici lanciati è in costante aumento, questi vengono lanciati su infrastrutture cloud, strutture IT e persino dispositivi IoT; non è solo il numero di attacchi che aumenta con il passare del tempo, ma sono anche le tecniche di attacco che si evolvono, ne nascono di nuove e altre cambiano la loro natura di comportamento nel tempo.

Infatti, secondo quanto riportato nel Clusit Report 2022, nel 2021 gli attacchi in tutto il mondo sono aumentati del 10% rispetto all'anno precedente, e stanno diventando sempre più gravi. Le nuove modalità di attacco dimostrano che i cybercriminali sono sempre più sofisticati e in grado di fare rete con la criminalità organizzata. La gravità degli attacchi è in netto aumento. Nel 2021, infatti, il 79% degli attacchi rilevati ha avuto un impatto "elevato", rispetto al 50% dello scorso anno. La motivazione principale è la criminalità informatica, che rappresenta l'86% degli attacchi informatici, in aumento rispetto all'81% del 2020. Questi numeri dovrebbero far riflettere: le persone riguardo la vulnerabilità della loro privacy, le aziende riguardo vulnerabilità della loro proprietà intellettuale e del loro patrimonio informativo.

Nel linguaggio comune i criminali informatici sono chiamati hacker, ma il termine corretto è cracker. Infatti, l'hacker è colui che utilizza le proprie

competenze e conoscenze per esplorare e imparare senza causare danni a persone e aziende, mentre il cracker è colui che agisce illegalmente per causare danni o profitti.

Questo progetto di tesi mira a fornire una soluzione tramite dei modelli di Deep Learning per riconoscere e bloccare eventuali attacchi informatici di diversa natura. Dopo un'attenta analisi sulle soluzioni disponibili presenti ad oggi, i modelli utilizzati, ritenuti i migliori per lo scopo del lavoro di tesi, fanno riferimento a due diversi tipi di approcci:

- LSTM Neural Network;
- Random Forest Classifier.

Il primo ha consentito di ottenere un modello migliore in termini di accuracy ma molto più dispendioso in termini di capacità computazionale rispetto al secondo.

Il dataset utilizzato per la costruzione dei due modelli è stato l'UNSW-NB15, pubblicato nel 2015, in Australia, dall'Università del 'New South Wales', contenente diverse informazioni, utili alla classificazione dei diversi attacchi nelle varie categorie.

Per la valutazione dei modelli, oltre al valore di accuracy, per entrambe le soluzioni, è stata calcolata la confusion matrix unita ai valori di precision e recall. Tutti questi risultati hanno consentito di individuare il miglior modello, in grado di individuare i diversi tipi di attacchi.

Le soluzioni mostrate all'interno del lavoro di tesi, sono state ottenute mediante l'utilizzo di un PC con le seguenti caratteristiche:

- CPU: Intel Core i7-8750H - 2.20GHz;
- RAM: 16 GB;
- Sistema Operativo: Windows 11;
- IDE: PyCharm 2021.

Abstract

No one is safe on the web, cyber-attacks are becoming more frequent and companies, especially large companies, are under the eye of cybercriminals.

Moreover, cloud technologies have spread exponentially in the world, thus major concern come in play regarding the cyber security and the privacy of firms and even single persons that use services from cloud providers. The number of cyber-attacks launched are constantly increasing, these are launched on cloud infrastructures, IT structures and even IoT devices; it is not only the number of attacks that increases with the transit of time, but it is also the attack techniques that evolve, new ones born and other change their nature of behavior within the time [1]. In fact, according to what is stated in the Clusit 2022 Report, in 2021, attacks around the world increased by 10% from the previous year, and they are getting more serious. New attack modes show that cyber criminals are increasingly sophisticated and able to network with organized crime. The severity of attacks has been increasing sharply. In fact, in 2021, 79% of detected attacks had a "high" impact, up from 50% last year. The main motivation was cybercrime, accounting for 86% of cyber-attacks, up from 81% in 2020 [12]. These numbers should make us think: people about the vulnerability of their privacy, companies on the vulnerability of their intellectual property and their information assets. Cybercriminals in common parlance are called hackers but the correct term is cracker. In fact, the hacker is the one who uses their skills and knowledge to explore and learn without causing harm to people and companies, while the cracker is the one who acts illegally to cause damage or profit.

This thesis project aims to provide a solution via Deep Learning model to recognize and block eventually cyber-attack of different natures.

Index

Introduction

Cloud computing performs several functionalities, and one of the most important functionalities is the storage and processing of data or information. With day-by-day enhancement of technology, cloud has been overburdened, and to address this issue, the concept of fog computing has been introduced.

Fog computing is an extension of the properties of cloud computing to the network's edge and additionally overcomes its limitations [2]. Despite the growing fame of fog services, assuring the security and privacy of data is still a big challenge. Distributed denial of service (DDoS) attack is a well-known threat among the security concerns and an important research challenge when talking particularly about security of data in fog computing environment.

In the meantime, the "Internet of Things" (IoT) vision has evolved and is coming to reality. The IoT involves several billions of diverse devices inter-connected by 2020 [3] vast amounts of quickly emerging/versatile data (i.e., "big data"), and numerous services. Connected devices can be sensors, computers, smart phones, and any other device or object that can be connected, monitored, or actuated. Devices are connected to the Internet, as well as with each other, via heterogeneous access networks. Services aim at leading to a smart, sustainable, and inclusive society and economy. In the light of the issues discussed, the success of the IoT services can only be achieved if they are attributed with ubiquitous accessibility, reliability, high performance, efficiency, and scalability [4] – [6].

Heterogeneous devices and interoperating connections can complicate the management of IoT systems, while many devices have resource and computing power limitations. These factors and their combinations sometimes complicate detection of abnormal IoT devices behaviour. To respond to security threats, it is necessary to apply modernized methods and tools for analysing a large number of events. These events are contained in network traffic packets, system logs and other data formats. In turn, data science can help identify patterns and correlate security events to improve IoT security. Great technological advances in machine learning are provided, which are opening up many possibilities for solving current and future IoT security challenges. Machine learning and, in particular, deep learning are currently powerful technologies for detecting attacks and detecting abnormal behaviour of smart devices.

The aim of this thesis is to provide a solution able to recognize different cyber-attack techniques at application level, which is increasingly difficult to detect because botnets tend to get confused with various legitimate users by using a comparison between Random Forest Machine Learning model and a deep learning model that combines the strength of a Convolutional Neural Network (CNN) and a bidirectional Long-Short-Term Memory (LSTM) neural network. [1]

1. State of the Art

1.1 Background

In computing, a denial-of-service attack (DoS attack) is a cyber-attack in which the perpetrator seeks to make a machine or network resource unavailable to its intended users by temporarily or indefinitely disrupting services of a host connected to the Internet. DoS attack is typically accomplished by flooding the targeted machine or resource with superfluous

requests to overload systems and prevent some or all legitimate requests from being fulfilled (US-CERT, 2013).

In a distributed denial-of-service attack (DDoS attack), the incoming traffic flooding the victim originates from many different sources. This effectively makes it impossible to stop the attack simply by blocking a single source. In this work, a solution is provided able to recognize the following 10 cyber-attack classes briefly described in the next sections: Normal, Analysis, Backdoor, DoS, Exploit, Fuzzers, Generic, Reconnaissance, Shellcode and Worms.

1.2 Related Work

After searching available literature, very limited works have been noticed to be done on the DDoS attack using logs file or based on the study of the behaviour of users and bots, through a User Behaviour Analytics (UBA) solution by using Long Short-Term Memory (LSTM).

A Naïve Bayes classifier [7] is deployed to classify packets into normal and attacked traffic. Several features and characteristics of packets in CAIDA dataset are used for analysis. CAIDA dataset is very old to tackle current version of App-DDoS attack.

RPV (Real-time Frequency Vector) [8] model with the real time characterization of traffic identifies App-DDoS attack including flash crowd by analysing the entropy of this attack. A defence infrastructure is also designed consisting of a detection system, traffic filter and headend sensor. It was claimed that this filter can mitigate the App-DDoS attack and allow a normal user to access the server. [1]

A model based on information gain and machine learning technique [9] named as random forest classifier, achieves the objective of detection of Denial-of-service attack. Optimal features are selected from available 41 features from the NSLKDD dataset using information gain. But NSl-KDD dataset doesn’t provide updated data as compared to today’s scenario and this model doesn’t provide information about selected features which are used for detecting this attack.

A statistical one-dimensional access matrix (ODAM) [10] model is proposed to check behaviour of DDoS attack, and which proposes an algorithm to detect this attack at application layer. This method differentiates DDoS attack and flash crowd in busy time and decreases false positive rate at significant level. [1]

1.3 Neural Networks solutions

Neural networks represent deep learning using artificial intelligence. Certain application scenarios are too heavy or out of scope for traditional machine learning algorithms to handle. As they are commonly known, Neural Network pitches in such scenarios filling the gap.

Artificial neural networks are inspired by the biological neurons within the human body which activate under certain circumstances resulting in a related action performed by the body in response. Artificial neural nets consist of various layers of interconnected artificial neurons powered by activation functions that help in switching them ON/OFF. Like traditional machine algorithms, here too, there are certain values that neural nets learn in the training phase.

Briefly, each neuron receives a multiplied version of inputs and random weights, which is then added with a static bias value (unique to each neuron layer); this is then passed to an appropriate activation function which decides the final value to be given out of the neuron. Once the output is generated from the final neural net layer, loss function (input vs output) is calculated, and backpropagation is performed where the weights are adjusted to make the loss minimum. Finding optimal values of weights is what the overall operation focuses around.

1.3.1 Feed Forward Neural Networks

It is the simplest form of neural networks where input data travels in one direction only, passing through artificial neural nodes and exiting through output nodes. Where hidden layers may or may not be present, input and output layers are present there. Based on this, they can be further classified as a single-layered or multi-layered feed-forward neural network.

Number of layers depends on the complexity of the function. It has unidirectional forward propagation but no backward propagation. Weights are static here. An activation function is fed by inputs which are multiplied by weights. To do so, classifying activation function or step activation function is used. For example: The neuron is activated if it is above threshold (usually 0), and the neuron produces 1 as an output. The neuron is not activated if it is below threshold (usually 0) which is considered as -1. They are simple to maintain and are equipped with to deal with data which contains a lot of noise.

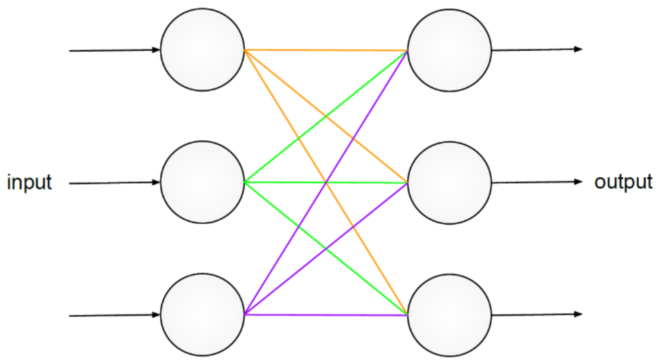


Fig. 1 – FNN Architecture

ADVANTAGES:

- Less complex, easy to design and maintain;
- Fast and speedy (One-way propagation);
- Highly responsive to noisy data.

DISADVANTAGES:

- Cannot be used for deep learning (due to absence of dense layers and back propagation).

1.3.2 Convolutional Neural Network

Convolution neural network contains a three-dimensional arrangement of neurons instead of the standard two-dimensional array. The first layer is called a convolutional layer. Each neuron in the convolutional layer only processes the information from a small part of the visual field. Input features are taken in batch-wise like a filter. The network understands the images in parts and can compute these operations multiple times to complete the full image processing. Processing involves conversion of the image from RGB or HSI scale to grey scale. Furthering the changes in the pixel value will help to detect the edges and images can be classified into different categories.

Propagation is unidirectional where CNN contains one or more convolutional layers followed by pooling and bidirectional where the output of convolution layer goes to a fully connected neural network for classifying the images as shown in the below diagram. Filters are used to extract certain parts of the image. In MLP the inputs are multiplied with weights and fed to the activation function. Convolution uses RELU and MLP uses nonlinear activation function followed by SoftMax. Convolution neural networks show very effective results in image and video recognition, semantic parsing, and paraphrase detection.

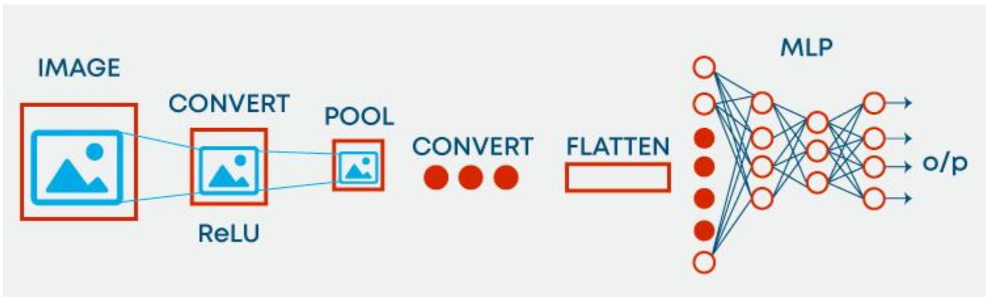


Fig. 2 – CNN Architecture

ADVANTAGES:

- Used for deep learning with few parameters;
- Less parameters to learn as compared to fully connected layer.

DISADVANTAGES:

- Comparatively complex to design and maintain;
- Comparatively slow (depends on the number of hidden layers).

1.3.3 Recurrent Neural Network

Designed to save the output of a layer, Recurrent Neural Network is fed back to the input to help in predicting the outcome of the layer. The first layer is typically a feed forward neural network followed by recurrent neural network layer where some information it had in the previous time-step is remembered by a memory function. Forward propagation is implemented in this case. It stores information required for its future use. If the prediction is wrong, the learning rate is employed to make small changes. Hence, making it gradually increase towards making the right prediction during the backpropagation.

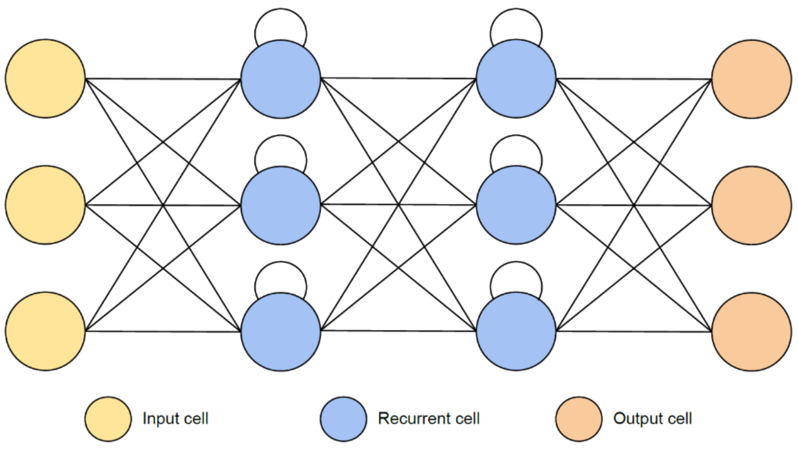


Fig. 3 – RNN Architecture

ADVANTAGES:

- Model sequential data where each sample can be assumed to be dependent on historical ones;
- Used with convolution layers to extend the pixel effectiveness.

DISADVANTAGES:

- Gradient vanishing and exploding problems;
- Training recurrent neural nets could be a difficult task;
- Difficult to process long sequential data using ReLU as an activation function.

1.3.4 LSTM (Long Short-Term Memory) Neural Network

LSTM networks are a type of RNN that uses special units in addition to standard units. LSTM units include a 'memory cell' that can maintain information in memory for long periods of time. A set of gates is used to control when information enters the memory when it's output, and when it's forgotten. There are three types of gates:

- Input gate
- Output gate
- Forget gate.

Input gate decides how many information from the last sample will be kept in memory; the output gate regulates the amount of data passed to the next layer, and forget gates control the tearing rate of memory stored. This architecture lets them learn longer-term dependencies

This is one of the implementations of LSTM cells:

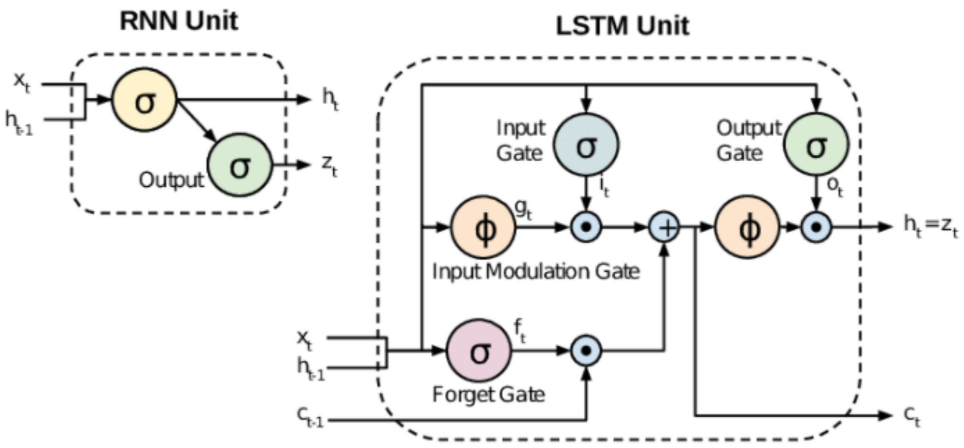


Fig. 4 – LSTM Implementation

1.4 LSTM configurations and evaluations

For our purpose, the most suitable neural network is LSTM, because as mentioned earlier, LSTM units include a 'memory cell' that can maintain information in memory for long periods of time, and this allows it to work well with time dependent sequential data, as in the present case.

The LSTM model is built via ‘Keras’ library an open-source Python library which run on ‘Tensorflow’.

Different configuration has been tasted and a comparison of the results, based on different metrics, has been done to evaluate the best model in terms of multiple KPIs. All the configurations are trained with a 90/10 training test split validation set which means the 90% of the dataset has been used to train the model and the remaining 10% has been used to validate the model. The whole process has been repeated 10 times shuffling the data between training and test set. The model has been trained with multiple configurations for each neural network mentioned above: with 1 hidden layer, 2 hidden layers then for each of them by changing the number of hidden nodes, from 32 nodes to 64 and then 128 nodes. The dropout rate has been changed too: none, one set 0.1 and the other with dropout rate set at 0.2.

The dropout rate represents the possibility of shutting down a neuron in the network due to avoid overfitting problem and for quick response in recurrent neural network.

Model Type	LSTM with no hidden layers	LSTM with 1 hidden layer (LSTM-1)	LSTM with 2 hidden layers (LSTM-2)	LSTM with 3 hidden layers (LSTM-3)
N° of hidden neurons = 32	87.67 %	91.33 %	94.68 %	93.67 %
N° of hidden neurons = 64	87.96 %	96.98 %	95.67 %	97.45 %
N° of hidden neurons = 128	89.88 %	96.45 %	95.89 %	97.21 %
Dropout = 0.0	89.88 %	96.45 %	95.89 %	93.29 %
Dropout = 0.1	90.13 %	91.57 %	97.39 %	96.78 %
Dropout = 0.2	90.98 %	92.89 %	98.88 %	98.34 %

Table 1 - LSTM Neural networks performances - validation accuracy percentage

From the results summarized in the above table can be observed that LSTM with two hidden layers, 128 hidden neurons and a dropout rate set at 0.2 is outperforming in terms of accuracy on the test set the remaining configurations. [1]

1.5 Random Forest solution

Random forests are ensemble of trees such that each tree is constructed on bootstrapped sample of the original training data. To classify a new object from an input vector, the input vector will be put down each of the trees in the forest. Each tree gives a vote to indicate the tree's decision about the class of the object. The forest chooses the classification having the most votes over all the trees in the forest. Each tree of the forest is grown as follows:

1. Let the number of examples in the original training data is N. Draw a bootstrap sample of size N from the original training data. This sample will be a new training dataset for growing the tree. Data which are in the original training data but not in the bootstrap sample are called out-of-bag data.
2. Let the total number of input features in the original training data be M. On this bootstrap sample data, only m attributes are chosen at random for each tree where m < M. The attributes from this set creates the best possible split at each node of the tree. The value of m should be constant during the growing of the forest.

The **accuracy** (strength) of the individual trees and the correlation between the trees in the forest determine the error rate of the forest. While increasing the correlation increases the forest error rate, increasing the accuracy of the individual tree decreases the forest error rate. Both strength and correlation are dependent on m. That is reducing m reduces both the correlation and the strength.

Compared to a single decision tree algorithm, random forests run efficiently on large datasets with a better accuracy. It can handle nominal data and does not over-fit. Final decision for classification of test data is done by majority votes from predictions of the ensemble of trees. [14]

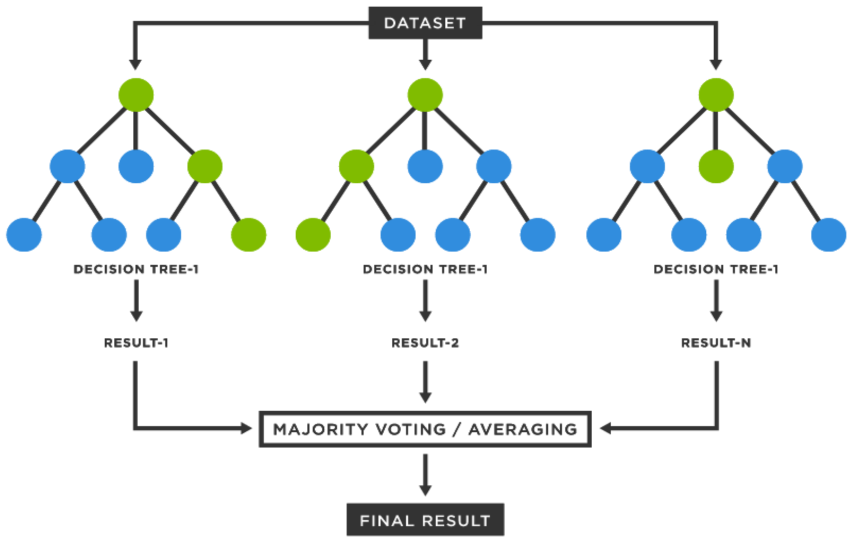


Fig. 5 - Random Forest

ADVANTAGES:

- It reduces overfitting in decision trees and helps to improve the accuracy
- It is flexible to both classification and regression problems
- It works well with both categorical and continuous values
- It automates missing values present in the data
- Normalising of data is not required as it uses a rule-based approach.

DISADVANTAGES:

- It requires much computational power as well as resources as it builds numerous trees to combine their outputs.
- It also requires much time for training as it combines a lot of decision trees to determine the class.
- Due to the ensemble of decision trees, it also suffers interpretability and fails to determine the significance of each variable.

2. Dataset

2.1 Feature Description

One of the major research challenges in this field is the unavailability of a comprehensive network-based dataset which can reflect modern network traffic scenarios, vast varieties of low footprint intrusions and depth structured information about the network traffic. Evaluating network intrusion detection systems research efforts, KDD98, KDDCUP99 and NSLKDD benchmark data sets were generated a decade ago. However, numerous current studies showed that for the current network threat environment, these datasets do not inclusively reflect network traffic and modern low footprint attacks. The UNSW-NB15 dataset was published by University of New South Wales, Australia in 2015 which marked the limitations of KDD98 and KDD99 data sets including the fact that these datasets do not include modern low footprint attacks. The raw network packets of the UNSW-NB 15 data set are created by the IXIA PerfectStorm tool in the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) for generating a hybrid of real modern normal activities and synthetic contemporary attack behaviours. Tcpdump tool is utilised to capture 100 GB of the raw traffic (e.g., Pcap files). This data set has nine families of attacks, namely, Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms. The Argus, Bro-IDS tools are utilised, and twelve algorithms are developed to totally generate 49 features with the class label. These features are described in *UNSW-NB15_freatures.csv* file. The total number of records is two million. This dataset has a hybrid of the real modern normal and the contemporary synthesized attack activities of the network traffic. Existing and novel methods are utilized to generate the features of the UNSWNB15 data set. The dataset includes a variety of packet-based features and flow-based features. The packet-based features describe the headers and the payload of each packet. In the flow-based features only connected packets send to the network are considered to keep a low computational analysis during the dataset creation.

#	*Name*	*Type*	*Description*
1	*srcip*	string	Source IP address
2	*sport*	integer	Source port number
3	*dstip*	string	Destination IP address
4	*dsport*	integer	Destination port number
5	*proto*	string	Transaction protocol

Table 2 – UNSW-NB15 Flow features

The flow-based features and the basic feature, as can be observed from the above and below tables are based on the direction of the packets.

#	*Name*	*Type*	*Description*
6	state	string	Indicates to the state and its dependent protocol, e.g. ACC, CLO, CON, ECO, ECR, FIN, INT, MAS, …
7	dur	float	Record total duration
8	sbytes	integer	Source to destination transaction bytes
9	dbytes	integer	Destination to source transaction bytes
10	sttl	integer	Source to destination time to live value
11	dttl	integer	Destination to source time to live value
12	sloss	integer	Source packets retransmitted or dropped
13	dloss	integer	Destination packets retransmitted or dropped

14	service	string	http, ftp, smtp, ssh, dns, ftp-data, irc and (-) if not much used service
15	sload	float	Source bits per second
16	dload	float	Destination bits per second
17	spkts	integer	Source to destination packet count
18	dpkts	integer	Destination to source packet count

Table 3 – UNSW-NB15 Basic features

#	*Name*	*Type*	*Description*
19	swin	integer	Source TCP window advertisement value
20	dwin	integer	Destination TCP window advertisement value
21	stcpb	integer	Source TCP base sequence number
22	dtcpb	integer	Destination TCP base sequence number
23	smeansz	integer	Mean of the flow packet size transmitted by the src
24	dmeansz	integer	Mean of the flow packet size transmitted by the dst
25	trans_depth	integer	Represents the pipelined depth into the connection of http request/response transaction
26	res_bdy_len	integer	Actual uncompressed content size of the data transferred from the server's http service.

Table 4 – UNSW-NB15 Content features

#	*Name*	*Type*	*Description*
27	sjit	float	Source jitter (mSec)
28	djit	float	Destination jitter (mSec)
29	stime	timestamp	Record start time
30	ltime	timestamp	record last time
31	sintpkt	float	Source interpacket arrival time (mSec)

32	dintpkt	float	Destination interpacket arrival time (mSec)
33	tcprtt	float	TCP connection setup round-trip time The sum of ’synack’ and ’ackdat’ of the TCP
34	synack	float	TCP connection setup time, the time between the SYN and the SYN_ACK packets.
35	ackdat	float	TCP connection setup time, the time between the SYN_ACK and the ACK packets

Table 5 – UNSW-NB15 Time features

This first 35 features represent the integrated gathered information from data packets: most of the features are generated from the header of the packets and include information on time, content, base information, and flow information of the packets.

#	***Name***	***Type***	***Description***
36	is_sm_ips_ports	Boolean	If source (1) equals to destination (3) IP addresses and port numbers (2)(4) are equal, this variable takes value 1 else 0
37	ct_state_ttl	integer	No. for each state (6) according to specific range of values for source/destination time to live (10) (11).
38	ct_flw_http_mthd	integer	No. of flows that has methods such as Get and Post in http service.

39	is_ftp_login	boolean	If the ftp session is accessed by user and password, then 1 else 0.
40	ct_ftp_cmd	integer	No of flows that has a command in ftp session.

Table 6 – UNSW-NB15 Genal purpose features

The features in the above table are considered general purpose features, each feature has it owns purpose according to the defense point of view.

#	***Name***	***Type***	***Description***
41	ct_srv_src	integer	No. of connections that contain the same service (14) and source address (1) in 100 connections according to the last time (26).
42	ct_srv_dst	integer	No. of connections that contain the same service (14) and destination address (3) in 100 connections according to the last time (26).
43	ct_dst_ltm	integer	No. of connections of the same destination address (3) in 100 connections according to the last time (26).
44	ct_src_ ltm	integer	No. of connections of the same source address (1) in 100 connections according to the last time (26).
45	ct_src_dport_ltm	integer	No of connections of the same source address (1) and the destination port (4) in 100 connections according to the last time (26).
46	ct_dst_sport_ltm	integer	No of connections of the same destination address (3) and the source port (2) in 100 connections according to the last time (26).

47	ct_dst_src_ltm	integer	No of connections of the same source (1) and the destination (3) address in in 100 connections according to the last time (26).

Table 7 – UNSW-NB15 Connection features

The features from 41 to 46 are labeled as connection features. This feature is needed in order to provide defense during attempt to connection scenario, the attackers can scan host for example on time per minute or one time per hour. The features from 36 to 47 are intended to sort accordingly with the last time feature, to capture similar characteristics of the connection records for each 100 connections sequentially ordered. At the end we can describe the two labelled features that we want to predict with this dataset.

#	***Name***	***Type***	***Description***
48	attack_cat	string	The name of each attack category. In this data set, nine categories (e.g., Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms)
49	label	boolean	0 for normal traffic and 1 for attack records

Table 8 – UNSW-NB15 Labelled features

This data set is labelled as listed in the table above: “attack_cat” is a multi-categorical feature useful in order to predict the type of attack that the network is suffering, and “label” column: for each record set to 0 if the record is normal and 1 if the record is attack one. The values of the attack categories label are listed in the table below.

Type of record	*N° of record*	*Description*
Normal	2,218,761	Natural transaction data.
Fuzzers	24,246	Attempting to cause a program or network suspended by feeding it the randomly generated data
Analysis	2,677	It contains different attacks of port scan, spam and html files penetrations
Backdoors	2,329	A technique in which a system security mechanism is bypassed stealthily to access a computer or its data
DoS	16,353	A malicious attempt to make a server or a network resource unavailable to users, usually by temporarily interrupting or suspending the services of a host connected to the Internet
Exploits	44,525	The attacker knows of a security problem within an operating system or a piece of software and leverages that knowledge by exploiting the vulnerability.
Generic	215,481	A technique works against all block ciphers (with a given block and key size), without consideration about the structure of the block-cipher.
Reconnaissance	13,987	Contains all Strikes that can simulate attacks that gather information.

Shellcode	1,511	A small piece of code used as the payload in the exploitation of software vulnerability.
Worms	174	Attacker replicates itself to spread to other computers. Often, it uses a computer network to spread itself, relying on security failures on the target computer to access it.

Table 9 – UNSW-NB15 "attack_cat" categorical values, sorted in descending order by number of records available in the dataset

UNSWNB15 is created by establishing the synthetic environment at the UNSW cyber security lab. The key utilized IXIA tool, has provided the capability to generate a modern representative of the real modern normal and the synthetical abnormal network traffic in the synthetic environment. UNSW-NB15 represents nine major families of attacks by utilizing the IXIA PerfectStorm tool. There are 49 features that have been developed using Argus, Bro-IDS tools and twelve algorithms which cover characteristics of network packets. In contrast the existing benchmark data sets such as KDD98, KDDCUP99 and NSLKDD, realized a limited number of attacks and information of packets which are outdated [11].

3. Data preprocessing

Multiple pre-processing techniques have been applied to the dataset in order to make it usable by the LSTM neural network in the most efficient way possible, in this chapter we will deal in detail with all the pre-processing steps that were necessary for the dataset preparation.

In particular, data preprocessing, a component of data preparation, describes any type of processing performed on raw data to prepare it for another data processing procedure. It has traditionally been an important preliminary step for the data mining process. More recently, data preprocessing techniques have been adapted for training machine learning models and AI models and for running inferences against them.

Data preprocessing transforms the data into a format that is more easily and effectively processed in data mining, machine learning and other data science tasks. The techniques are generally used at the earliest stages of the machine learning and AI development pipeline to ensure accurate results.

Virtually any type of data analysis, data science or AI development requires some type of data preprocessing to provide reliable, precise and robust results for enterprise applications. Real-world data is messy and is often created, processed and stored by a variety of humans, business processes and applications. As a result, a data set may be missing individual fields, contain manual input errors, or have duplicate data or different names to describe the same thing. Humans can often identify and rectify these problems in the data they use in the line of business, but data used to train machine learning or deep learning algorithms needs to be automatically preprocessed.

Machine learning and deep learning algorithms work best when data is presented in a format that highlights the relevant aspects required to solve a problem. Feature engineering practices that involve data wrangling, data transformation, data reduction, feature selection and feature scaling help restructure raw data into a form suited for particular types of algorithms. This can significantly reduce the processing power and time required to train a new machine learning or AI algorithm or run an inference against it.

The steps used in data preprocessing include the following:

- Data profiling.
- Data cleaning
- Data reduction
- Data transformation
- Data enrichment
- Data validation

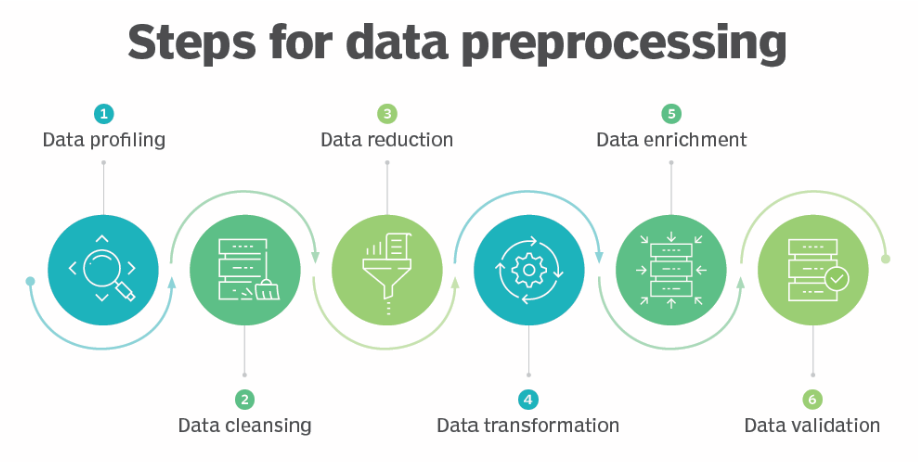
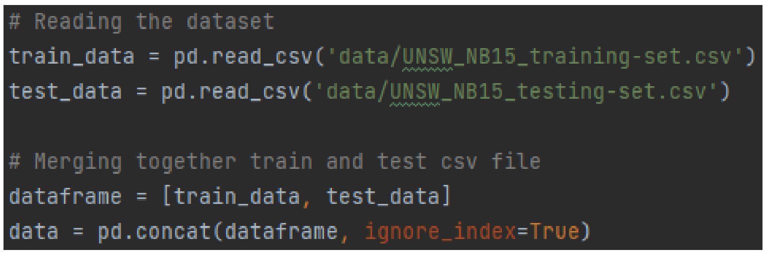


Fig. 6 – Steps for data preprocessing

3.1 Data profiling

Data profiling is the process of examining, analysing, and reviewing data to collect statistics about its quality. It starts with a survey of existing data and its characteristics. Data scientists identify data sets that are pertinent to the problem at hand, inventory its significant attributes, and form a hypothesis of features that might be relevant for the proposed analytics or machine learning task. They also relate data sources to the relevant business concepts and consider which preprocessing libraries could be used.

In the present case, starting from the reading of train and test set csv files, have been done some steps: in particular, they have been merged and mixed, in order to increase the data dimensionality in terms of number of records available.



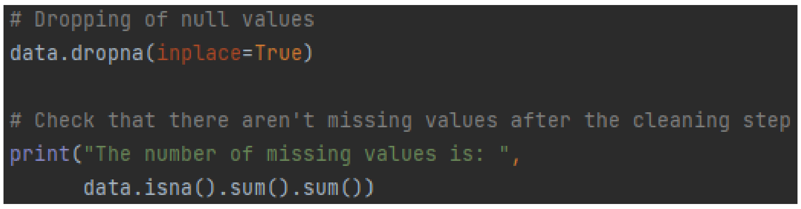
```
# Reading the dataset
train_data = pd.read_csv('data/UNSW_NB15_training-set.csv')
test_data = pd.read_csv('data/UNSW_NB15_testing-set.csv')

# Merging together train and test csv file
dataframe = [train_data, test_data]
data = pd.concat(dataframe, ignore_index=True)
```

Fig. 7 – Merge train and test data

3.2 Data cleaning

The aim of data cleaning is to find the easiest way to rectify quality issues, such as eliminating bad data, filling in missing data or otherwise ensuring the raw data is suitable for feature engineering.

In particular, the data cleaning step, has be done by checking the null values and then dropping it in order to clean the dataset.



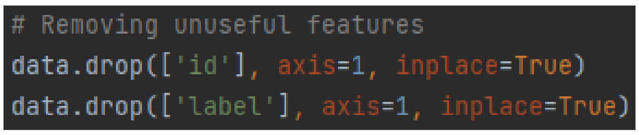
```
# Dropping of null values
data.dropna(inplace=True)

# Check that there aren't missing values after the cleaning step
print("The number of missing values is: ",
      data.isna().sum().sum())
```

Fig. 8 – Drop of missing values

3.3 Data reduction

Raw data sets often include redundant data that arise from characterizing phenomena in different ways or data that is not relevant to a particular ML, AI, or analytics task. Data reduction uses techniques like principal component analysis to transform the raw data into a simpler form suitable for particular use cases.

In fact, some features have been removed, because not useful for the computation. Specifically, the features ‘*id’* and ’*label*’ have been dropped from the dataset.



```
# Removing unuseful features
data.drop(['id'], axis=1, inplace=True)
data.drop(['label'], axis=1, inplace=True)
```

Fig. 9 – Drop of unhelpful features

3.4 Data transformation

About this step, data scientists think about how different aspects of the data need to be organized to make the most sense for the goal. This could include things like structuring unstructured data, combining salient variables when it makes sense or identifying important ranges to focus on, and normalizing the data that allows to transform them in a way that makes it easier for algorithms to tease apart a meaningful relationship between variables.

In the present case, about data transformation has been done One Hot Encoding to transform some categorical features in numerical one, followed by Z-Score Normalizzation.

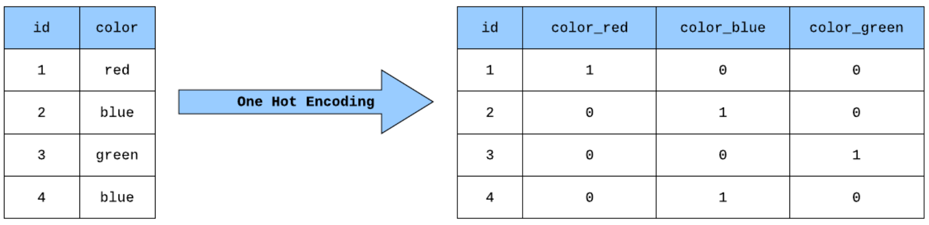
3.4.1 One Hot Encoding

In particular, three categorical features: *'proto'*, *'state'* and '*service'* has been One Hot Encoded from the dataset. Many machine learning algorithms cannot operate on label data directly. They require all input variables and output variables to be numeric. This means that categorical data must be converted to numerical form. If the categorical variable is an output variable, you may also want to convert predictions by the model back into a categorical form in order to present them or use them in some application [16].

Let's now analyse how the transformation of the feature "*service*" takes place. Originally the categorical feature has this set of unique values in the dataset as follows:

- *"-"*
- *"dhcp"*
- *"dns"*
- *"ftp"*
- *"ftp-data"*
- *"http"*
- *"irc"*
- *"pop3"*
- *"radius"*
- *"smtp"*
- *"snmp"*
- *"ssh"*
- *"ssl"*

As previously discussed, this set of values for the categorical variable, are unusable by the neural network. The one hot encoding process so will drop the feature "*service*" and then will add, in this case, 13 new dummy boolean features, representing each unique values in the original feature "*service*". For each record of the dataset only one of this 13 new features will have value "1", the remaining 12 will have value "0". The feature with value set to "1" will indicate the type of connection used for the fixed record of the dataset making the feature understandable from the LSTM neural network. In order to achieve this process *getdummies* function of pandas python library has been used.



id	color
1	red
2	blue
3	green
4	blue

id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

Fig. 10 – *Example of One hot Encoding on the feature "color"*

One-hot Encoding is chosen over label encoder since label encoder will produce multiple number in the same column, the model might misunderstand these values to be in a particular order and this will impact the classification [1].

3.4.2 Normalizzation

The next step of the pre-processing phase consists in normalizing the numerical features. Normalization is rescaling the data into a particular range to reduce redundancy and improve training time of the model.

In this model has been used Z-Score Normalization to rescale the range of the data to [0,1] in all the numerical features.

The formula for Z-score normalization is:

$$x_{new} = \frac{(x_i - \mu)}{\sigma}$$

In the formula above:

- x_i: is the original i-th sample
- μ: is the mean over all the data
- σ: is the standard deviation of data

Z-score normalization is a strategy of normalizing data that avoids the outlier issue, which instead occurs in Min-Max Normalizzation. For every feature, the minimum value of that feature gets transformed into a 0, the maximum value gets transformed into a 1, and every other value gets transformed into a decimal between 0 and 1.

The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values.

This is a graphic example, to better understand how Z-Score works on data:

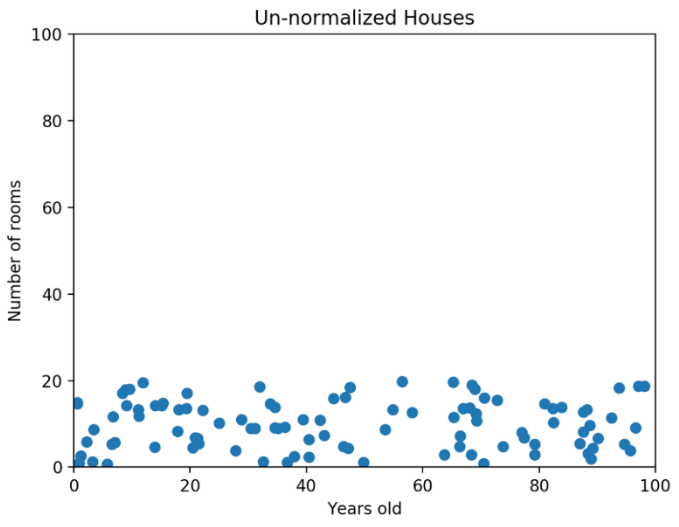


Fig. *11 - Example of feature values sparsity before Z-Score normalization*

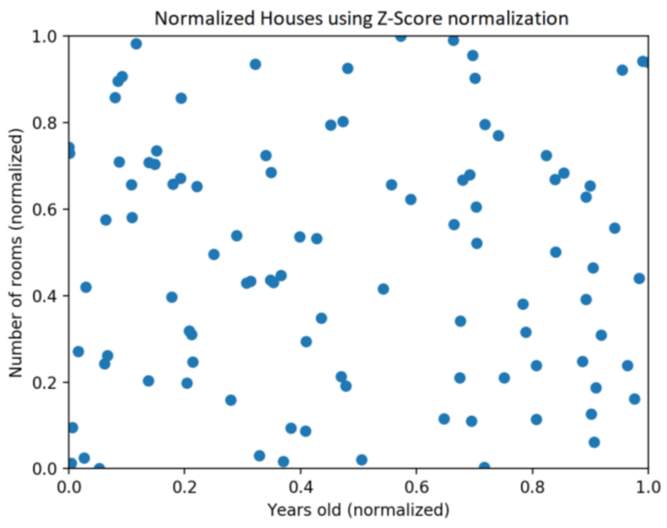


Fig. *12 - Example of feature values sparsity after Z-Score normalization*

Thanks to normalization every datapoint have the same scale so each feature is equally important.

3.5 Data enrichment

In this step, data scientists apply the various feature engineering libraries to the data to effect the desired transformations. The result should be a data set organized to achieve the optimal balance between the training time for a new model and the required compute.

In the present case, the number of samples of minority class "Worms" in UNSW-NB15 dataset is 174 and this is very few compared to the total number of samples 257.673. This bias in the training dataset can influence the machine learning algorithm, leading to ignore the minority class entirely. This is a problem as it is typically the minority class on which predictions are most important.

One approach to addressing the problem of class imbalance is to randomly resample the training dataset. The two main approaches to randomly resampling an imbalanced dataset are to delete examples from the majority class, called under sampling, or to duplicate samples from the minority class, called oversampling.

3.5.1 Random Over Sampling

For our purpose, random over sampling has been implemented on the less recurrent feature to predict, which is ‘*Worms’*.

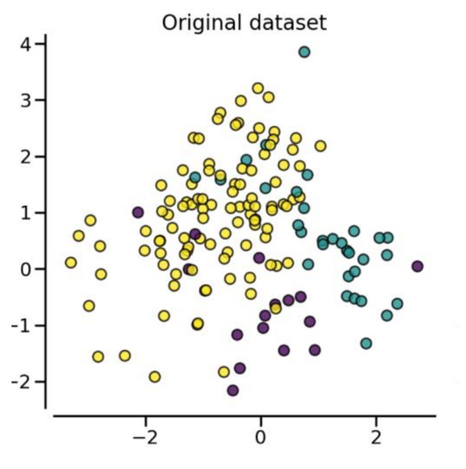


Fig. 13- Example of dataset diversity before Oversampling application

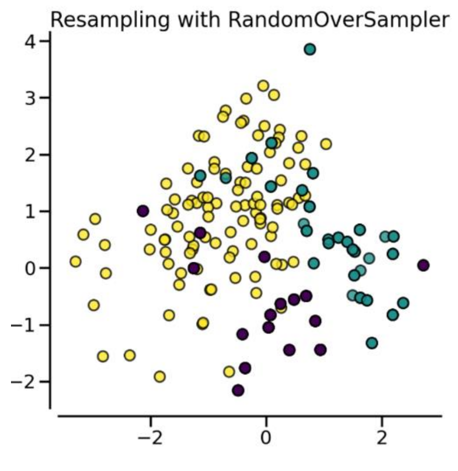
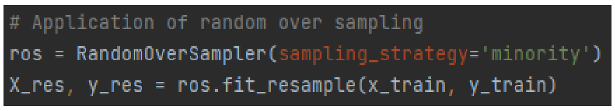


Fig. 14 - Example of dataset diversity after Oversampling application on the minority feature

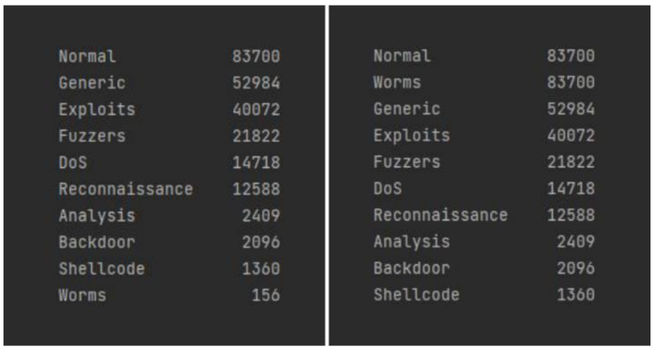
Thus, the last step of the project work pre-processing phase is Oversampling. Random Oversampling duplicates data points randomly from the minority features, this reduces the data imbalance and improves prediction accuracy of minority class. *RandomOverSampler* class of *imblearn.oversampling* python library is used for oversampling with ‘*minority’* as parameter/strategy.



```
# Application of random over sampling
ros = RandomOverSampler(sampling_strategy='minority')
X_res, y_res = ros.fit_resample(x_train, y_train)
```

Fig.15 – Python application of Random over sampling

Hence random oversampling technique is applied only on the training set of UNSW-NB15 and that lead to a significant increase in accuracy and detection rate of the class “Worms”, considered as a minority in the dataset Image 14 show the Prediction label row count before (a) and after (b) performing Oversampling. The dataset is now ready to be fed into the LSTM neural network.



```
Normal            83700
Generic           52984
Exploits          40072
Fuzzers           21822
DoS               14718
Reconnaissance    12588
Analysis           2409
Backdoor           2096
Shellcode          1360
Worms               156
```

```
Normal            83700
Worms             83700
Generic           52984
Exploits          40072
Fuzzers           21822
DoS               14718
Reconnaissance    12588
Analysis           2409
Backdoor           2096
Shellcode          1360
```

Fig.16 - Prediction label row count before(a) and after (b) performing Oversampling.

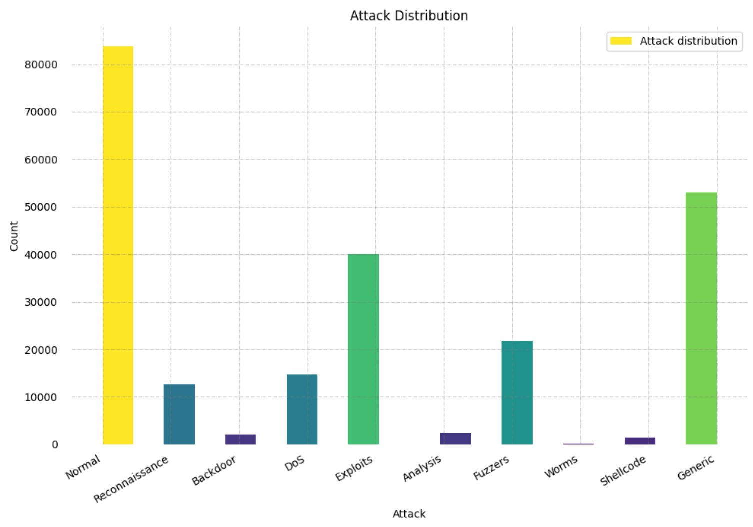


Fig.17 – Histogram of Prediction label row count before Oversampling

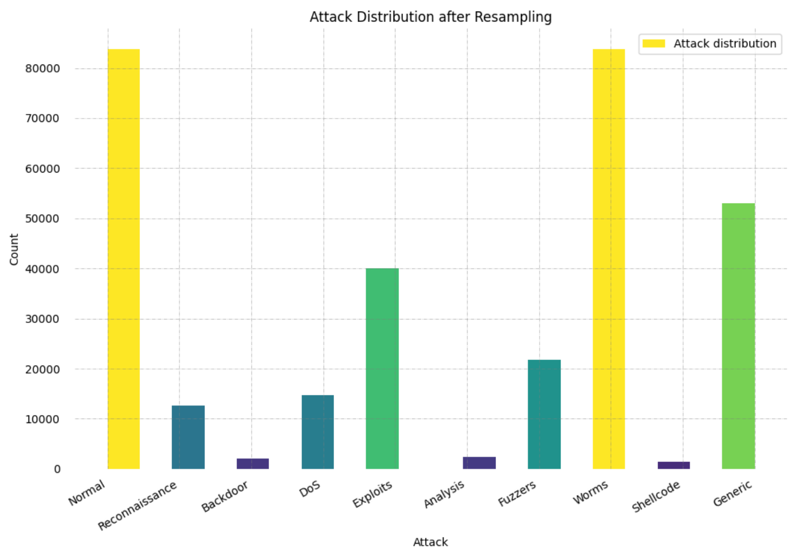


Fig. 18 – Histogram of Prediction label row count after Oversampling.

3.6 Data validation

At this stage, the data is split into two sets. The first set is used to train a machine learning or deep learning model. The second set is the testing data that is used to gauge the accuracy and robustness of the resulting model. This second step helps identify any problems in the hypothesis used in the cleaning and feature engineering of the data.

If the data scientists are satisfied with the results, they can push the preprocessing task to a data engineer who figures out how to scale it for production. If not, the data scientists can go back and make changes to the way they implemented the data cleansing and feature engineering steps.

4. Model Implementation

4.1 Introduction

The aim of this work is to propose a deep learning model to compare two different solutions.

- The first one has been done by combining the strength of a Convolutional Neural Network (CNN) and a **bidirectional LSTM** neural network that will give to the model the ability to learn from spatial and temporal features coming from the dataset.

- The second one, has been done by using the **random forest classifier** that reduces overfitting in decision trees and helps to improve the accuracy.

To train and test this model a publicly available dataset called UNSW-NB15 [11] has been used. This important dataset contains a categorical prediction variable beyond a boolean one. In fact, in addition to indicating whether the type of traffic on the network is malicious or not, this dataset describes in detail in 10 different classes the type of attack itself:

- Normal (no attack)
- DoS
- Exploits
- Generic
- Reconnaissance
- Worms
- Shellcode
- Analysis
- Backdoor
- Fuzzers

The model will be able to predict what type of attack the network is undergoing in a multi-class classification model.

4.2 Model Choice

4.2.1 Bi-LSTM Model Choice

For this thesis work an hierarchical model has been built up by combining layers of 1D-CNN and Bidirectional LSTM.

Convolutional Neural Network (CNN) is used to learn the spatial/high-level features of a dataset and the Bi-LSTM layers (sub-category of RNN) to learn the long time-range temporal features of the data and combine these to predict attacks.

Traditionally, machine learning techniques such as Support Vector Machine (SVM), Random Forest and Adaptive Boosting have been often used by researchers for constructing Network Intrusion Detection classifiers. Researchers have also used K-Mean clustering for efficient classification, but they have always turned out to be weak because of high False Positive Rate (FPR), overfitting and with lower accuracy on classes having less percentage of data available as compared to the classes in sufficient numbers.

The reason being, the traditional machine learning approaches concentrate upon learning feature importance, feature availability and dimensionality reduction techniques to find the most optimum correlation between data points that seem to have the most amount of influence upon the end-result, while completely overlooking the importance of correlation between the features and consider the time-steps to predict the best possible result. This led to the adoption of deep learning approaches to resolve the lacking points citied above. In fact, thanks to the '*timestamp'* column available in the dataset, makes the recurrent neural network (RNN) a default choice for approaching the problem solution. [1]

The RNN contains internal feedback loop useful to store time information associations, end up forming an acyclic graph as a result. In particular, in RNN each component shares the same weights. Generally, the advantage of recurrent networks is that they share weights for each position of the input vector. Also, a model can process sequences with different lengths by sharing the weights. Another advantage is that it reduces the number of parameters (weights) that the network needs to learn.

The basic principle in recurrent networks is that the input vector and some information from the previous step (generally a vector) are used to calculate the output and information passed to the next step. In general, the formulas used to calculate the output values in each step are called units (blocks).

Thus, for the simplest recurrent network, one block can be defined by the relation:

$$a^{<t>} = f_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

$$\hat{y}^{<t>} = f_2\left(W_{ya}a^{<t>} + b_y\right)$$

In which, $x^{<t>}$ is a vector or a number that's part of an input sequence, t indicates the step for calculating recurrent relations, W_{aa}, W_{ax}, W_{ya}, b_a and b_y are weight matrices and vectors with given dimension, and f_1 and f_2 are activation functions. Usually, for f_1, is used *tanh* or *ReLU*, while for f_2, since it calculates the output value, is used *sigmoid* or *softmax* as activation functions.

In RNN the backpropagation mechanism leads to the problem of vanishing and exploding the gradients: the features learned at the very start of the network start to have the very least amount of effect on the result of the model. To solve this problem **LSTM** (Long Short-Term Memory) cells are used in order to solve it. In addition, this type of network is better for maintaining long-range connections, recognizing the relationship between values at the beginning and end of a sequence.

The LSTM model introduces expressions, in particular, gates. In fact, there are three types of gates:

- *forget gate*: controls how much information the memory cell will receive from the memory cell from the previous step
- *update (input) gate*: decides whether the memory cell will be updated. Also, it controls how much information the current memory cell will receive from a potentially new memory cell.
- *output gate*: controls the value of the next hidden state

Mathematically, we define an LSTM block as:

$$\Gamma_u = \sigma(W_{uu}a^{<t-1>} + W_{ux}x^{<t>} + b_u)$$

$$\Gamma_f = \sigma(W_{ff}a^{<t-1>} + W_{fx}x^{<t>} + b_f)$$

$$\Gamma_o = \sigma(W_{oo}a^{<t-1>} + W_{ox}x^{<t>} + b_o)$$

$$\hat{c}^{<t>} = tanh(W_{cc}a^{<t-1>} + W_{cx}x^{<t>} + b_c)$$

$$c^{<t>} = \Gamma_u \odot \hat{c}^{<t>} + \Gamma_f \odot c^{<t-1>}$$

$$a^{<t>} = \Gamma_o \odot tanh(c^{<t>})$$

Where, W and b are weight matrices and vectors, t is the current iteration of the recurrent network, Γ_u is the update gate, Γ_f is the forget gate, Γ_o is the output gate, $\hat{c}^{<t>}$ is the potential value of the memory cell, $c^{<t>}$ is the current value of the memory cell, and $a^{<t>}$ is the output value or hidden state. The architecture of the LSTM block can be shown as:

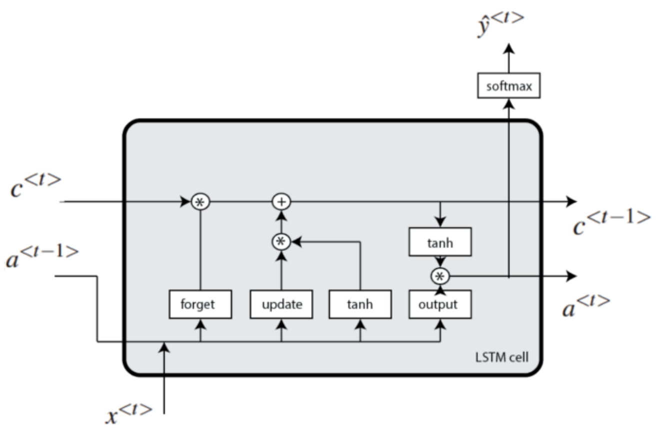


Fig. 19 – LSTM Architecture

The idea of merging LSTM (RNN) and 1D-CNN let the CNN to be used for learning spatial features by increasing the number of kernels which makes learning coarse grained, at the start of the network, and fine-grained, at the end of the network. The **Bidirectional LSTM** is a type of LSTM where the learning data are feed into to the model from the start to its end and the backwards again from the end to the start as well, this allow the network a better learning at every timestamp of the data, helping time related feature to be better digested through the network.

In summary, **Bi-LSTM** adds one more LSTM layer, which reverses the direction of information flow. Briefly, it means that the input sequence flows backward in the additional LSTM layer. Then we combine the outputs from both LSTM layers in several ways, such as average, sum, multiplication, or concatenation. The architecture of Bi - LSTM is showed below:

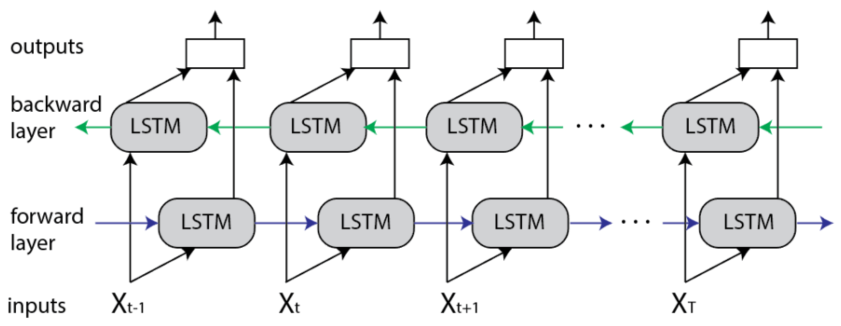


Fig. 20 –Bi-LSTM Architecture

This type of architecture has many advantages in real-world problems. The main reason is that every component of an input sequence has information from both the past and present. For this reason, BiLSTM can produce a more meaningful output, combining LSTM layers from both directions. [15]

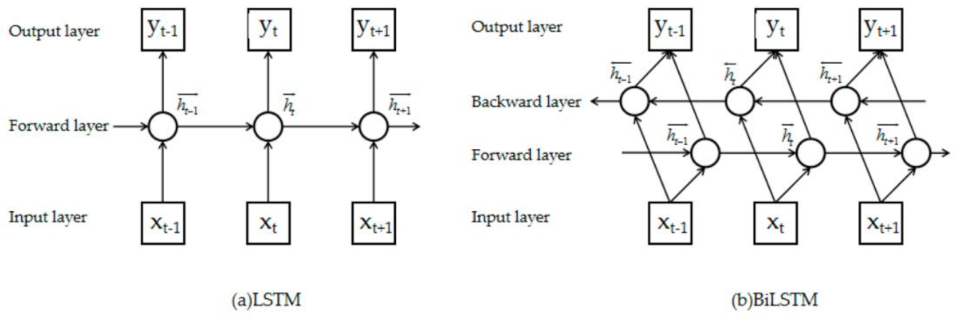


Fig. 21 - Difference between LSTM layer (a) and Bidirectional LSTM layer (b)

4.2.2 Random Forest Model Choice

For this thesis work, to compare the results obtained by the previous model (Bi-LSTM), Random Forest Classifier has been built up that contains an ensemble of Decision Trees.

“A decision tree is a classification as well as a regression technique. It works great when it comes to taking decisions on data by creating branches from a root, which are essentially the conditions present in the data, and providing an output known as a leaf.” [17]

To sum up, a random forest is a collection of Decision Trees, Each Tree independently makes a prediction, the values are then max voted, for the classification tasks, as in the present case, to arrive at the final value.

The strength of this model lies in creating different trees with different sub-features from the features. The Features selected for each tree is Random, so the trees do not get deep and are focused only on the set of features.

Finally, when they are put together, we create an ensemble of Decision Trees that provides a well-learned prediction.

Thus, this model can be useful for our purpose because provides an accuracy which is very high; moreover, it is particularly efficient with large dataset as in the present case and it does not overfit with more features. For all these reasons Random Forest Classifier seems to be a good choice to deal with this thesis work.

4.3 Model Architecture

4.3.1 Bi-LSTM Model's Architecture

Considering the observations made in the previous subchapter, the architecture of the model's architecture starts with a **1-D Convolutional layer,** followed by a **Max Polygon layer,** both useful to share parameters, spatial arrangement, and local perception characteristics. In particular:

- *Parameters sharing* brings into the network the ability to reduce the set of parameters, brining feature extraction in order to reduce the computational power needed by the model to run.
- *Spatial arrangement* brings an arrangement of feature in a sparse matrix that brings into the network the ability to learn from the features correlations.

So, after the first layer of 1-D CNN allows the network to learn from time series data, the Max pooling layer is useful to perform the discretization of parameters in order to recognize the relevant features resulting in reduced training time and prevention of overfitting.

The Max Polygon layer is followed by a **Batch normalization layer** that enables the normalization of the parameters in the middle of the architecture to reduce the training time. Furthermore, we found the first **Bidirectional LSTM layer** able to learn from both forward and backwards time series data. It is followed by a **Reshape layer** that enables to feed data into the Bi-LSTM layer both forward and backwards, meaning input and output have the same shape. This process provides a boost in the training time with better learning of features resulting in greater precision for a long spanning time-series data.

Next, we found in the network another **Max pooling layer** followed by a **Bach normalization layer** needed respectively for dismiss the least relevant features and normalize the output data of the previous intermediate layer to boost performance and decrease training times. After this we found the next **Bidirectional LSTM layer**: the first Bi-LSTM layer have a kernel dimension of 128 neurons, instead, this one has 256 neurons. The reason behind this choice lies in the fact of exploiting the use of coarse grain to fine grained learning to better understand the correlation of long-range time dependent features learnt by the first 1-D CNN layer which provides for better extraction of features and faster training times.

Next, in the architecture we can observe a **dropout layer** with a dropout rate set to 0.6 in order to prevent the neural network from an overfitting state, even if we have max pooling layer in between. The reason behind this lies in the fact that using both CNN and RNN brings an higher probability of overfitting and consequentially in bad performance on the testing set. The idea behind the dropout is to randomly (with 60% of probability) turn off some single unit within a layer, in this way we force the other units to learn every time some different information, thus enhancing the final performance of the layer. Finally, at the end we found a **dense layer** which serve as an output layer to the last output layer with activation function Softmax useful for multiclass classification.

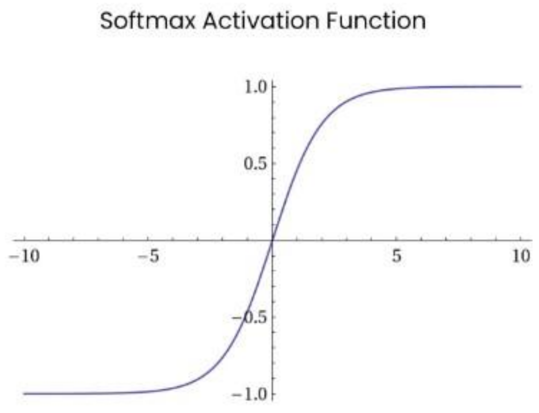
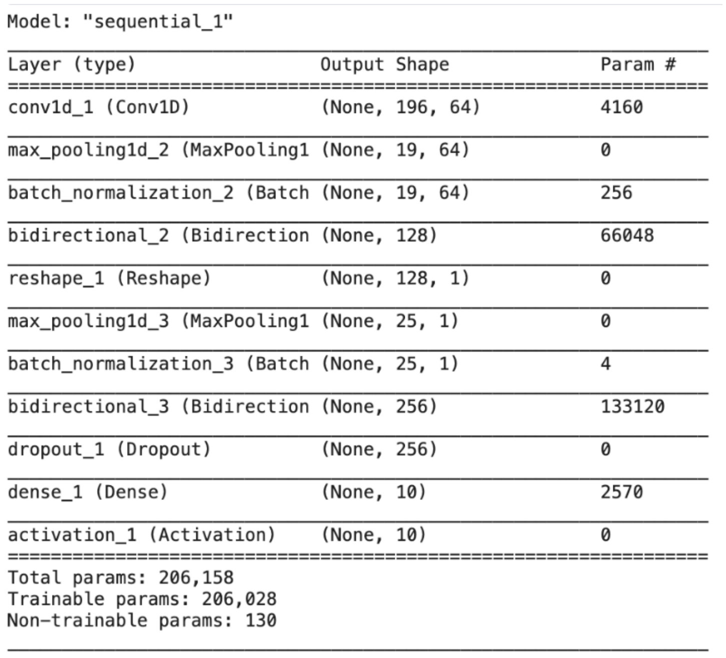


Fig. 22 – Softmax Activation Function

For each record, and consequentially prediction, the network will provide a probability for the fixed record to belong to each one of the prediction classes, in this case the different kind of attack types or benevolent traffic. The Softmax function will set to 0 the lowest probability to belong to fixed classes and bring to 1 the higher probably prediction class belonging to each record. In the figure below is shown the model architecture, using the *model.summary()* function from the Keras python library:



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 196, 64)           4160
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 19, 64)            0
_________________________________________________________________
batch_normalization_2 (Batch (None, 19, 64)            256
_________________________________________________________________
bidirectional_2 (Bidirection (None, 128)               66048
_________________________________________________________________
reshape_1 (Reshape)          (None, 128, 1)            0
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 25, 1)             0
_________________________________________________________________
batch_normalization_3 (Batch (None, 25, 1)             4
_________________________________________________________________
bidirectional_3 (Bidirection (None, 256)               133120
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                2570
_________________________________________________________________
activation_1 (Activation)    (None, 10)                0
=================================================================
Total params: 206,158
Trainable params: 206,028
Non-trainable params: 130
_________________________________________________________________
```

Fig. 23 – Model's Architecture snapshot

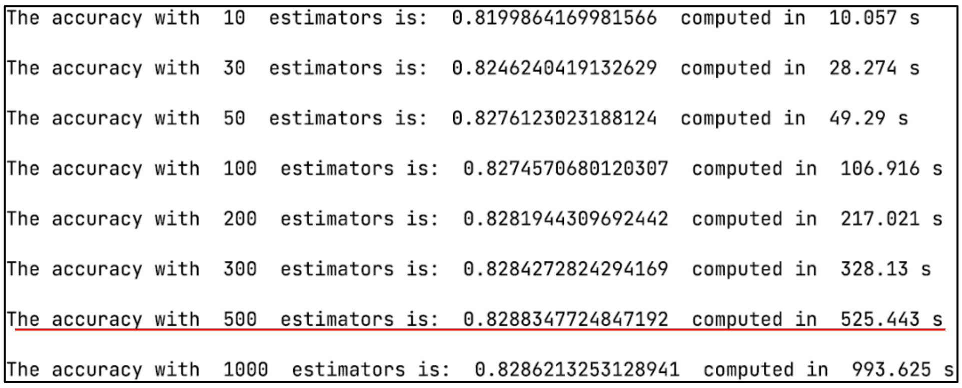
4.3.2 Random Forest Model’s Architecture

Considering the observations made in the previous subchapter to choose the best architecture for the Random Forest classifier model the main challenge has been the right selection of ‘*n_estimators*’ number.

It represents the number of trees needed to build before taking the maximum voting or averages of predictions. Higher number of trees gives better performance but makes the code slower.

For these reasons to choose the optimum number of *n_estimators*, has been computed the accuracy starting from the random forest algorithm applied n-times, by changing the values of estimators from 10 to 1000.

The results have shown in the figure below:



```
The accuracy with  10  estimators is:  0.8199864169981566  computed in  10.057 s

The accuracy with  30  estimators is:  0.8246240419132629  computed in  28.274 s

The accuracy with  50  estimators is:  0.8276123023188124  computed in  49.29 s

The accuracy with  100  estimators is:  0.8274570680120307  computed in  106.916 s

The accuracy with  200  estimators is:  0.8281944309692442  computed in  217.021 s

The accuracy with  300  estimators is:  0.8284272824294169  computed in  328.13 s

The accuracy with  500  estimators is:  0.8288347724847192  computed in  525.443 s

The accuracy with  1000  estimators is:  0.8286213253128941  computed in  993.625 s
```

Fig. 24 – Accuracies of Random Forest

Starting from the results as shown in the previous figure, the optimum number of estimators, which means, the optimum number of decision trees forming the random forest is **500** which provide an accuracy of **82.88%**.

4.4 Model Training

"A machine learning training model is a process in which a machine learning (ML) algorithm is fed with sufficient training data to learn from." [18]

Model training in machine language is the process of feeding an ML algorithm with data to help identify and learn good values for all attributes involved. The training model is used to run the input data through the algorithm to correlate the processed output against the sample output. The result from this correlation is used to modify the model.

This iterative process is called "model fitting". The accuracy of the training dataset or the validation dataset is critical for the precision of the model.

There are several types of machine learning models, of which the most common ones are supervised and unsupervised learning:

- *Supervised learning* is possible when the training data contains both the input and output values. Each set of data that has the inputs and the expected output is called a supervisory signal. The training is done based on the deviation of the processed result from the documented result when the inputs are fed into the model

- *Unsupervised learning* involves determining patterns in the data. Additional data is then used to fit patterns or clusters. This is also an iterative process that improves the accuracy based on the correlation to the expected patterns or clusters. There is no reference output dataset in this method.

In the thesis work, the model chosen to build the algorithm for the intrusion detection system regards supervised learning models. Therefore, an important step in model training is about the splitting of the dataset in train and test part.

4.4.1 Dataset Splitting strategies

There are different types of splitting strategy:

- **Holdout Splitting validation method**
- **K-Folds Cross Validation**
- **Random Subsampling validation method**

Holdout Splitting

This is the most classical technique, in which the dataset is splitted into the two parts of training and test set.

The training set is what the model is trained on, and the test set is used to see how well that model performs on unseen data. A common split when using the hold-out method is using 80% of data for training and the remaining 20% of the data for testing.

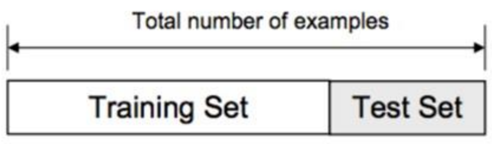


Fig. 25 – Holdout Splitting

K-Folds Cross Validation

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. In k-fold cross validation, the original sample is randomly partitioned into k equal sized subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k - 1 subsamples are used as training data. In the model training phase, cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data to overcome situations like overfitting.

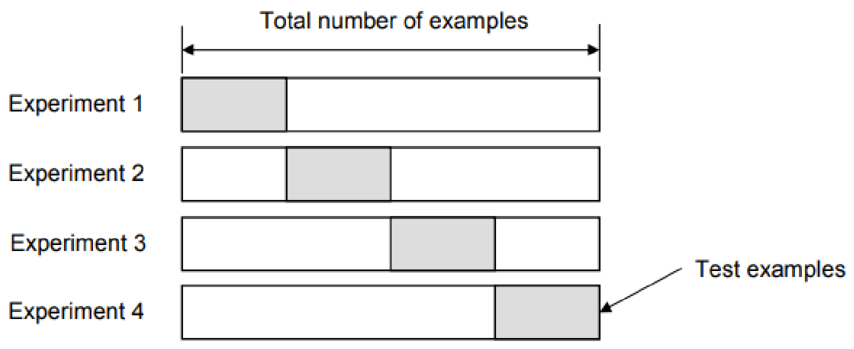


Fig. 26 – K-Folds Cross validation

Random subsampling validation method

Under this method data is randomly partitioned into dis-joint training and test sets multiple times means multiple sets of data are randomly chosen from the dataset and combined to form a test dataset while remaining data forms the training dataset.

The accuracies obtained from each partition are averaged and error rate of the model is the average of the error rate of each iteration. The advantage of random subsampling method is that it can be repeated an indefinite number of times. [19]

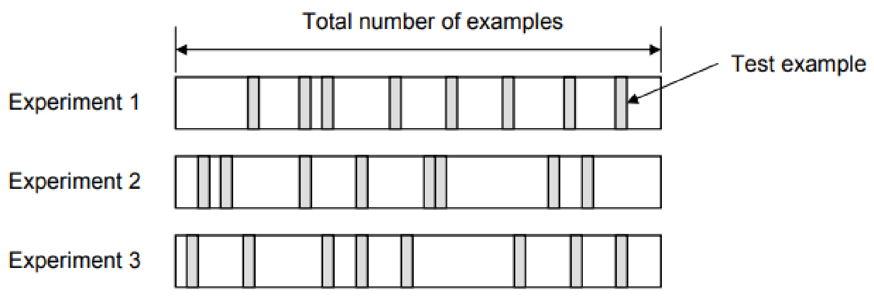


Fig. 27 – Random Subsampling

4.4.2 K-Fold Cross Validation

After analysing the different splitting strategies mentioned above, *K-Folds cross validation* was chosen because it avoids the problems of overfitting and underfitting the data, performing better in terms of evaluations than the other algorithms. So, the original dataset has been splitted randomly in k-subsets called folds of equal dimension. Of the k-folds created, only one folds will be used as a test set to validate the model, the remaining k-1 folds will be used to train the neural network model as a training set. This process is looped k-times to let every single fold to be used as a test set for validating the model. At the end the k results obtained by the model in terms of metrics (accuracy, loss, etc.) are averaged to evaluate the model in a fair way.

The advantage of this method consists in the fact that all the observation or records are used for both the training and the test phase. Furthermore, this approach generates test sets such that all contains the same distribution of classes, or as close as possible.

The number of k-folds selected that let us achieve the performance discussed in the next chapter are k = 10 and for each training the model is trained with a number of epochs equal to 10. Stratification ensures that each fold is a good representation of the whole dataset, this leads to parameter fine tuning and helps the model in classifying the attacks better.

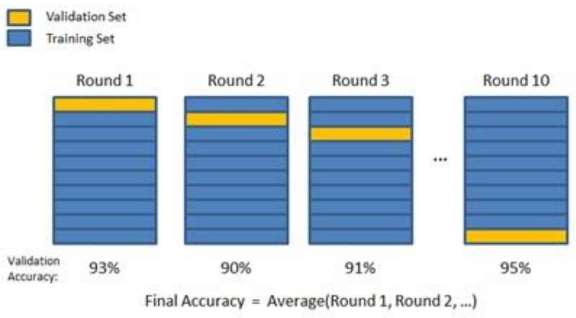


Fig 28 – Representation of how Cross validation works

5. Evaluation

The most important task in building any machine learning model is to evaluate its performance. Evaluation metrics are tied to machine learning tasks. There are different metrics for the tasks of classification and regression. Some metrics, like precision-recall, are useful for multiple tasks.

Classification and regression are examples of supervised learning, which constitutes a majority of machine learning applications. Using different metrics for performance evaluation, we should be able to improve our model's overall predictive power before we roll it out for production on unseen data. Without doing a proper evaluation of the Machine Learning model by using different evaluation metrics, and only depending on accuracy, can lead to a problem when the respective model is deployed on unseen data and may end in poor predictions.

5.1 Classification Metrics

Classification is about predicting the class labels given input data. In binary classification, there are only two possible output classes. In multi-class classification, more than two possible classes can be present as in the present case in which it is needed to predict different attack types.

There are many ways for measuring classification performance. Accuracy, confusion matrix, log-loss, and AUC-ROC are some of the most popular metrics. Precision-recall is a widely used metrics for classification problems.

In this thesis work the metrics used to evaluate the Machine Learning models are:

- Accuracy
- Confusion Matrix
- Precision
- Recall

5.2 Confusion Matrix

Confusion Matrix is a summary of predicted results in specific table layout that allows visualization of the performance measure of the machine learning model for a binary classification problem (2 classes) or multi-class classification problem (more than 2 classes) as in the present case in which we have 10 different predicted classes. As the name suggests, confusion matrix gives us a matrix as output and describes the complete performance of the model.

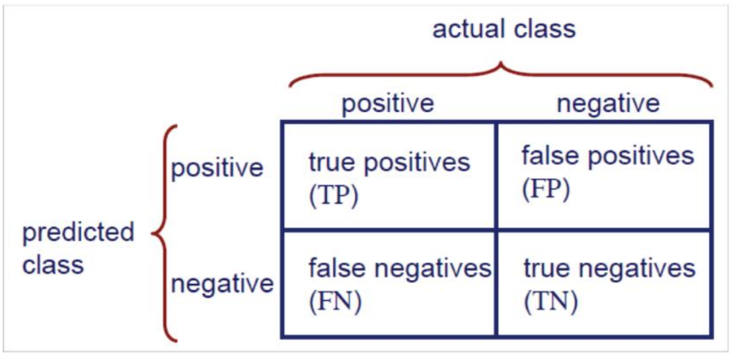


Fig. 29 – Example of binary confusion matrix

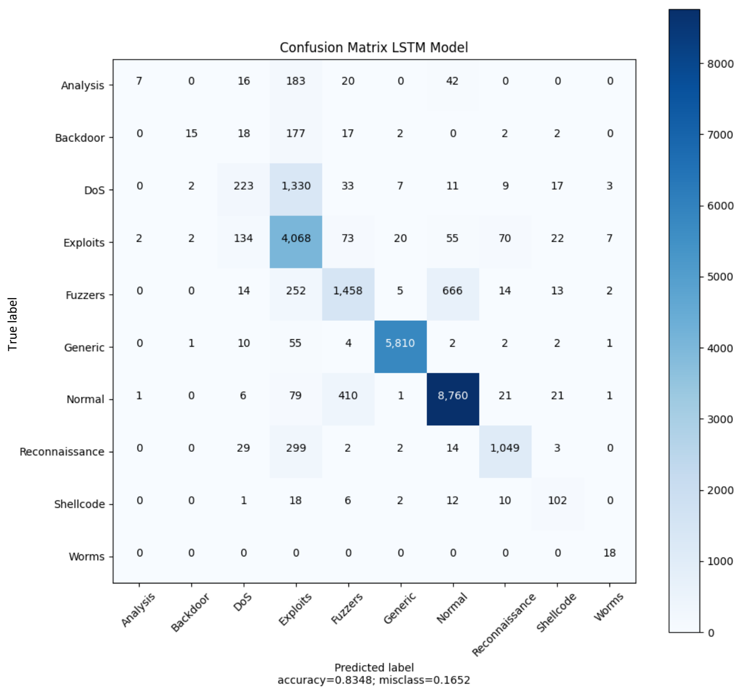
As is shown in the image above, there are 4 important terms:

- **TP (True Positives)**: The cases in which we predicted YES and the actual class was also YES;
- **TN (True Negatives):** The cases in which we predicted NO and the actual output was NO.

- **FP (False Positives):** The cases in which we predicted YES and the actual output was NO.
- **FN (False Negatives):** The cases in which we predicted NO and the actual output was YES. [20]

5.2.1 LSTM Confusion Matrix

The confusion matrix obtained starting from the LSTM model described in the previous chapters is shown in the figure below:


Confusion Matrix LSTM Model

True label \ Predicted label	Analysis	Backdoor	DoS	Exploits	Fuzzers	Generic	Normal	Reconnaissance	Shellcode	Worms
Analysis	7	0	16	183	20	0	42	0	0	0
Backdoor	0	15	18	177	17	2	0	2	2	0
DoS	0	2	223	1,330	33	7	11	9	17	3
Exploits	2	2	134	4,068	73	20	55	70	22	7
Fuzzers	0	0	14	252	1,458	5	666	14	13	2
Generic	0	1	10	55	4	5,810	2	2	2	1
Normal	1	0	6	79	410	1	8,760	21	21	1
Reconnaissance	0	0	29	299	2	2	14	1,049	3	0
Shellcode	0	0	1	18	6	2	12	10	102	0
Worms	0	0	0	0	0	0	0	0	0	18

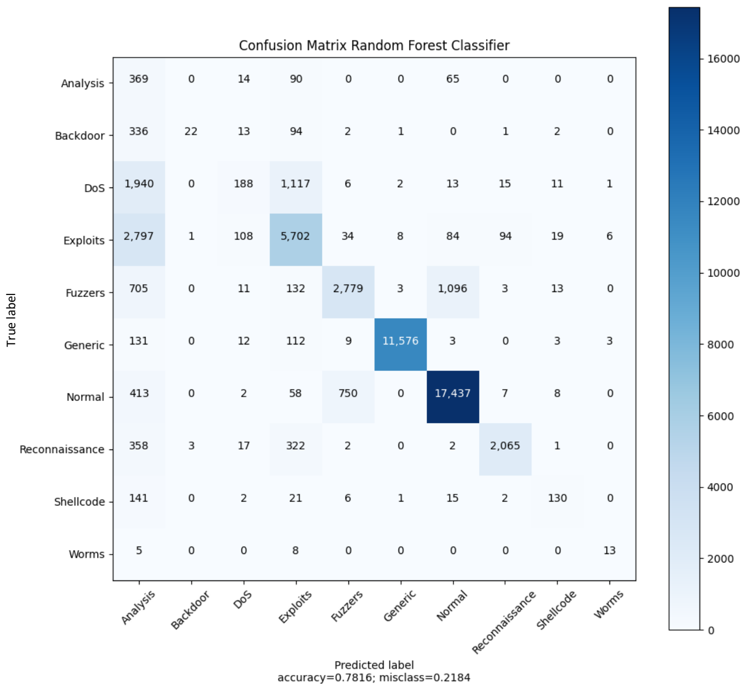
Fig. 30 – LSTM Confusion Matrix

As we can see in the confusion matrix shown above, in the main diagonal we find all the correctly predicted values, while the other values, omitting the values of the main diagonal, represent the wrong predictions. In the columns we can observe the False negative (FN) and in the rows we can observe the False positive (FP).

Therfore, looking at the obtained results, can be deduced that the LSTM built model provide an accuracy value of 0.8348 which means 83.48%. This result compared to [1] is about 1,2% better in terms of accuracy and of course in terms of miss classicifaction rate.

5.2.2 Random Forest Confusion Matrix

What was done for the LSTM model, has been also done for the Random Forest Classifier:



True label \ Predicted label	Analysis	Backdoor	DoS	Exploits	Fuzzers	Generic	Normal	Reconnaissance	Shellcode	Worms
Analysis	369	0	14	90	0	0	65	0	0	0
Backdoor	336	22	13	94	2	1	0	1	2	0
DoS	1,940	0	188	1,117	6	2	13	15	11	1
Exploits	2,797	1	108	5,702	34	8	84	94	19	6
Fuzzers	705	0	11	132	2,779	3	1,096	3	13	0
Generic	131	0	12	112	9	11,576	3	0	3	3
Normal	413	0	2	58	750	0	17,437	7	8	0
Reconnaissance	358	3	17	322	2	0	2	2,065	1	0
Shellcode	141	0	2	21	6	1	15	2	130	0
Worms	5	0	0	8	0	0	0	0	0	13

Fig. 31 – Random Forest Confusion Matrix

As can be seen in the upper figure, the accuracy obtained for the Random Forest Classifier built model is 0.7816, which means 78.16%.

By comparing this result with the previous one, can be easily seen that the LSTM model provides a better accuracy respect the Random Forest Classifier.

Therefore, based only on the accuracy value, we can easily think that the first model is better than the second to predict the different attack categories described in the original dataset. This is certainly true when the classes to predict are heterogeneous in the training dataset, but this is not the case. So, can be said that the first model provides a better accuracy than the second one and also with respect the model described in [1], but is needed to use other evaluation metrics starting from the concepts of TP, TN, FP, FN described before.

5.3 Accuracy

Accuracy is one of the metrics for evaluating classification models. It represents the fraction of predictions our model got right.

Formally, accuracy has the following definition:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{Number\ of\ correct\ precdictions}{Total\ number\ of\ predictions}$$

As it was said in the chapter 5.2.2, this metric works well only if there are equal number of samples belonging to each class.

For this reason, it is needed to use other evaluation metrics to better describe the results obtained starting from the built models. [matrix]

5.4 Precision and Recall

Precision and recall are two numbers which together are used to evaluate the performance of classification or information retrieval systems.

- **Precision** is defined as the fraction of relevant instances among all retrieved instances.

$$Precision = \frac{TP}{TP + FP}$$

- **Recall**, sometimes referred to as '*sensitivity*', is the fraction of retrieved instances among all relevant instances.

$$Recall = \frac{TP}{TP + FN}$$

A perfect classifier has precision and recall both equal to 1. Precision helps us understand how useful the results are. Recall helps us understand how complete the results are. [21]

Therefore, after a definition of Precision and Recall, it is possible to deep dive into the performance predictions of each one of the 10 classes.

5.4.1 LSTM Precision and Recall

In the figure below, are shown the results of Precision and Recall, obtained starting from the LSTM built model:

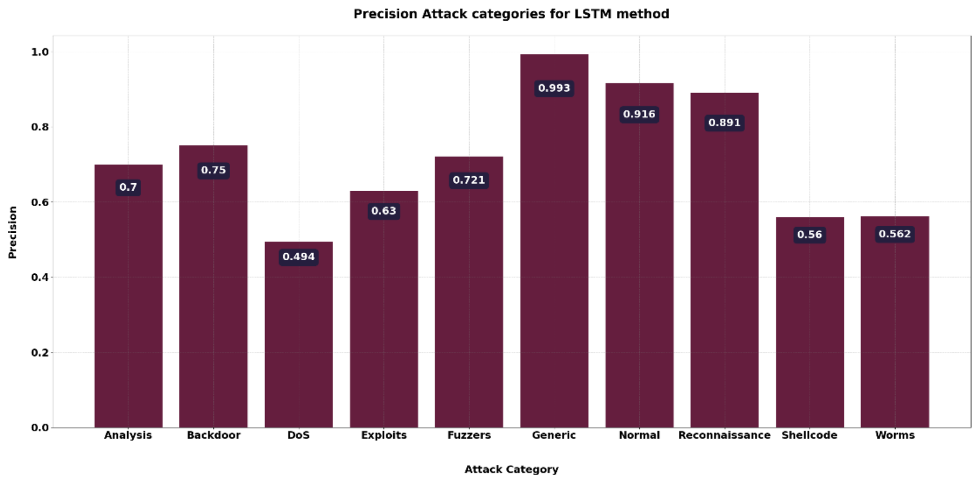


Fig 32 – Precision value of output classes for LSTM model

As we can see from the above chart, every time the model predicts the attack category ‘*Generic*’, 99,3% of the times the model is making a correct prediction. Extremely encouraging results were achieved in predicting the following classes respectively:

- ‘*Normal’* (91.6%),
- ‘*Reconnaissance’* (89.1%).

Good values are reached by the following class:

- ‘*Backdoor’* (75%),
- ‘*Fuzzers’* (72.1%),
- ‘*Analysis’* (70%),
- ‘*Exploits’* (63%).

The three classes:

- “Worms” (56.2%),
- “Shellcode” (56%),
- “DoS” (49.4%),

are the worst performing classes in terms of precision.

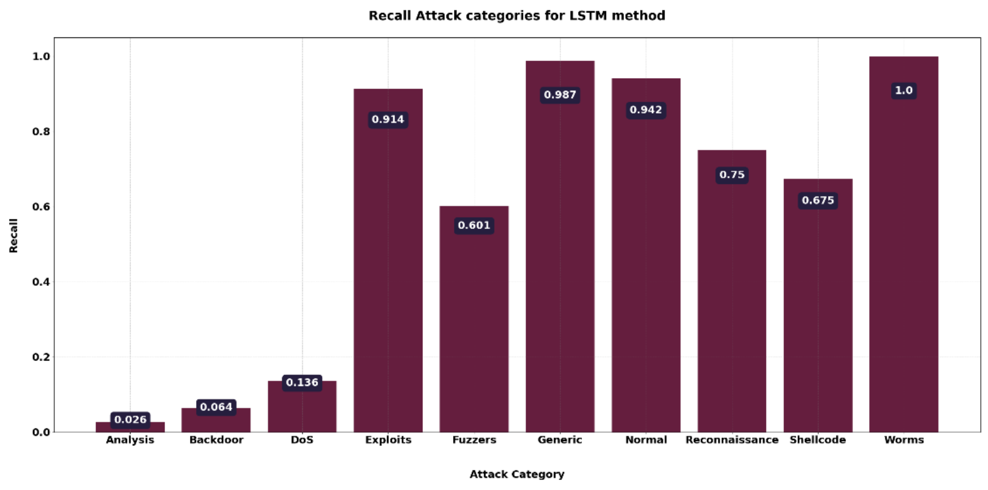


Fig 33 – Recall value of output classes for LSTM

From the chart above can be seen first of all that in term of recall the "Worms" class is reaching 100%, meaning every time there was a real worms attack going on, the model was able to predict it in a correct way; the root cause of this result can be found in the pre-processing phase of the dataset when the oversampling was applied to this class.

Encouraging results were achieved in predicting the following classes respectively:

- '*Generic'* (98.7%),
- '*Normal'* (94.2%),
- '*Exploits*' (91.4%),
- '*Reconnaissance*' (75%).

Sufficient good values are reached by these classes:

- '*Shellcode'* (67.5%),
- '*Fuzzers'* (60.1%).

Poor values are reached from the following three classes:

- '*DoS'* (13.6%),

- ‘*Backdoor*’ (6.4%),
- ‘*Analysis’* (2.6%)

5.4.2 Random Forest Precision and Recall

In the figure below, are shown the results of Precision and Recall, obtained starting from the Random Forest built model:

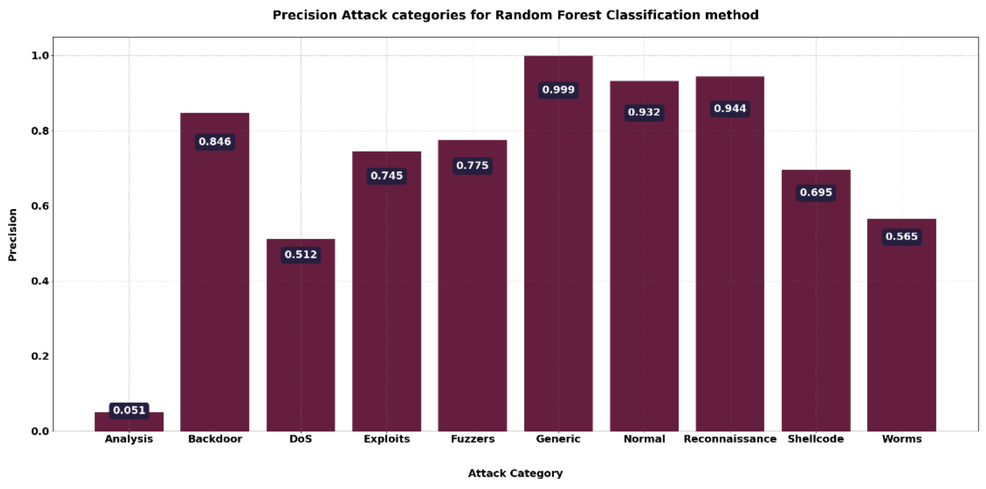


Fig 34 – Precision value of output classes for Random Forest model

As we can see from the above chart, every time the model predicts the attack category ‘*Generic*’, 99,9% of the times the model is making a correct prediction. Extremely encouraging results were achieved in predicting the following classes respectively:

- ‘*Reconnaissance’* (94.4%),
- ‘*Normal’* (91.6%),
- ‘*Backdoor’* (84.6%).

Good values are reached by the following class:

- ‘*Fuzzers’* (77.5%),

- ‘*Exploits’* (74.5%),
- ‘*Shellcode’* (69.5%).

Instead, the following three classes:

- ‘*Worms’* (56.5%),
- ‘*DoS’* (51.2%),
- ‘*Analysis’* (5.1%).

are the worst performing classes in terms of precision.

Comparing these results with those of the LSTM model, can be deduced that, except for *‘Analysis’* class, have been obtained better results, in terms of precision.

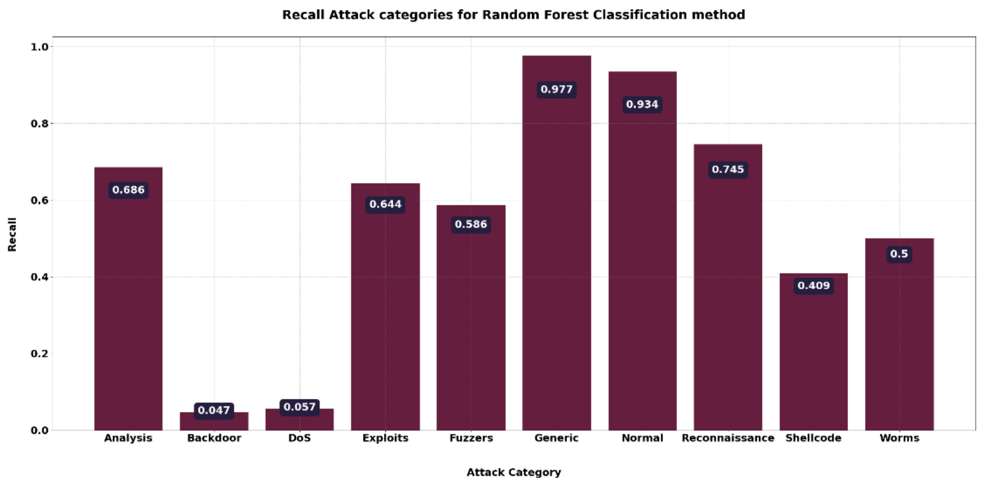


Fig 35 – Recall value of output classes for Random Forest

From the chart above can be seen first of all that in term of recall the ‘*Generic*’ class is reaching 99.7%, meaning that, almost every time there was a real Generic attack going on, the model was able to predict it in a correct way.

Encouraging results were achieved in predicting the following classes respectively:

- ‘*Normal’* (93.4%),
- ‘*Reconnaissance*’ (74.5%).

Sufficient good values are reached by these classes:

- ‘*Analysis’* (68.6%),
- ‘*Exploits*’ (64.4%),
- ‘*Fuzzers’* (58.6%),
- *‘Worms’* (50%),
- ‘*Shellcode’* (40.9%).

Poor values are reached from the following three classes:

- ‘*DoS’* (5.7%),
- ‘*Backdoor*’ (4.7%).

Comparing the recall results of Random Forest model with those of LSTM model, it is clear that the second one, provides better results in terms of recall values. Therfore, this results, leads us to compute a new evaluation metric, which is the F-score that can help to find the best model between the two built and well described in the previous chapters.

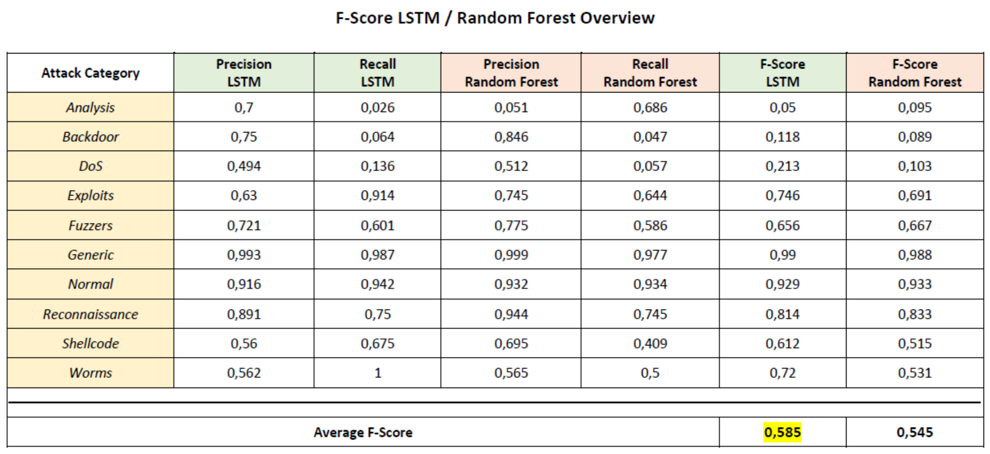
5.5 F-Score

Precision and recall are sometimes combined together into the F-score. Usually, precision and recall scores are given together and are not quoted individually. This is because it is easy to vary the sensitivity of a model to improve precision at the expense of recall, or vice versa.

If a single number is required to describe the performance of a model, the most convenient figure is the F-score, which is the harmonic mean of the precision and recall:

$$F - score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

This allows us to combine the precision and recall into a single number.



F-Score LSTM / Random Forest Overview

Attack Category	Precision LSTM	Recall LSTM	Precision Random Forest	Recall Random Forest	F-Score LSTM	F-Score Random Forest
Analysis	0,7	0,026	0,051	0,686	0,05	0,095
Backdoor	0,75	0,064	0,846	0,047	0,118	0,089
DoS	0,494	0,136	0,512	0,057	0,213	0,103
Exploits	0,63	0,914	0,745	0,644	0,746	0,691
Fuzzers	0,721	0,601	0,775	0,586	0,656	0,667
Generic	0,993	0,987	0,999	0,977	0,99	0,988
Normal	0,916	0,942	0,932	0,934	0,929	0,933
Reconnaissance	0,891	0,75	0,944	0,745	0,814	0,833
Shellcode	0,56	0,675	0,695	0,409	0,612	0,515
Worms	0,562	1	0,565	0,5	0,72	0,531
Average F-Score					**0,585**	**0,545**

Table 10 – F-score overview of LSTM and Random Forest Model

By looking at the F-score overview table, can be seen that in the most cases, LSTM built model provides better value with respect the Random Forest model, which means that the LSTM model has better performances in terms of attack categories classification with respect to the other model.

However, the average F-score computed value is better in LSTM case compared to the other one.

Therefore, as shown in the table above, in the LSTM model, given the precision results the following class: “Analysis”, “Generic”, “Normal”, “Reconnaissance”, “Backdoor”, “Fuzzers” and “Exploits” have reached good results when the model is making a prediction on one of this class. Instead, for the remaining class: “DoS”, “Shellcode” and “Worms” a larger dataset is needed in order to increase the precision in the prediction of this kind of attacks or simply removing this class, re-train the model without this mentioned class, for further increase the accuracy on the class which are already providing good precision scores.

The recall scores tell us basically that in a real environment, when real attacks data are fed into the model, the classes: “Generic”, “Normal”, “Exploits” and “Reconnaissance” will, with high probability, be correctly classified by the model; this leads to the following attack classes to be easily recognizable in a real use scenario without any consequences in making a wrong prediction.

6. Conclusions and Future Development

Since the use of cloud technologies has spread exponentially in the world, the use of Network intrusion detection system has become a field of vital importance in cyber security: with the endless growth of network traffic and the spread of new methods of attack, this type of technology has become a must that cloud environments cannot afford to ignore. The approach used in this project work based on machine learning and anomaly detection techniques highlights how the deep learning approach turns out to be the best weapon to identify and isolate this type of malicious attacks, surpassing in precision and accuracy approaches of pattern recognition and anomaly detection approaches more traditional like Support vector machine (SVM) or Decision tree (DT). The obtained values of accuracy, precision and recall let us understand on which classes the model can be further improved, increasing even more already excellent values of predictions and instead underline the classes in which the model need of being improved with training data more distributed in classes that are performing below the average.

Moreover, network topology changes made to the model (feature selection, number of neurons, leaning rate, time ranges, iterations, epochs, batch size, loss function, optimizer, training and validation methods, dataset dimensionality and distribution within each class, etc…) can be further adjusted in order to further improve the current results. Finally, the model could be integrated into software designed to fight the types of attacks with UBA solutions such as the open source OpenUBA project and then give an extra solution to the users that with this kind of tools try to defend their networks from such kind of malicious attack.

The source code of the thesis work is available at the following link:

https://github.com/pyteer9/Thesis.git

Figures Index

Table Index

References

Bibliography

[1] F. Nocera et al., "*Cyber-Attack Mitigation in Cloud-Fog Environment Using an Ensemble Machine Learning Model*", 2022 7th International Conference on Smart and Sustainable Technologies (SpliTech), 2022, pp. 1-6.

[2] N. Makitalo, F. Nocera, M. Mongiello et al., "*Architecting the web of things for the fog computing era*", IET Software, 2018.

[3] J. Holler and J. Arkko, "*Internet of things propels the networked society*", vol. 3, p. 2016, 2012.

[4] S. Sambangi and L. Gondi, "*A machine learning approach for ddos (distributed denial of service) attack detection using multiple linear regression*", vol. 63, no. 1, 2020. [Online]. Available: https://www.mdpi.com/2504-3900/63/1/51

[5] M. Mongiello, F. Nocera, A. Parchitelli, L. Patrono, P. Rametta, L. Riccardi, and I. Sergi, "*A smart iot-aware system for crisis scenario management*", Journal of Communications software and Systems, vol. 14, no. 1, pp. 91–98, 2018.

[6] G. Cavalera, R. C. Rosito, V. Lacasa, M. Mongiello, F. Nocera, L. Patrono, and I. Sergi, "*An innovative smart system based on iot technologies for fire and danger situations*", in 2019 4th International Conference on Smart and Sustainable Technologies (SpliTech). IEEE, 2019, pp. 1–6.

[7] N. A. Singh, K. J. Singh, and T. De, “*Distributed denial of service attack detection using naive bayes classifier through info gain feature selection*”, in Proceedings of the International Conference on Informatics and Analytics, 2016, pp. 1–9.

[8] W. Zhou, W. Jia, S. Wen, Y. Xiang, and W. Zhou, “*Detection and defense of application-layer ddos attacks in backbone web traffic*”, Future Generation Computer Systems, vol. 38, pp. 36–46, 2014.

[9] T. Pinto and Y. Sebastian, “*Detecting ddos attacks using a cascade of machine learning classifiers based on random forest and mlp-an*”, in Proceeding of 2021 IEEE Madras Section Conference (MASCON). IEEE, 2021, pp. 1–6.

[10] V. Kumar, H. Sharma et al., “*Detection and analysis of ddos attack at application layer using naïve bayes classifier,*”, Journal of Computer Engineering & Technology, vol. 9, no. 3, pp. 208–217, 2018.

[11] N. Moustafa and J. Slay, “*Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)*”, in Proceeding of Military Communications and Information Systems Conference (MilCIS), 2015, pp. 1–6.

[14] A. Tesfahun and D. L. Bhaskari, "Intrusion Detection Using Random Forests Classifier with SMOTE and Feature Reduction," 2013 International Conference on Cloud & Ubiquitous Computing & Emerging Technologies, 2013, pp. 127-132, doi: 10.1109/CUBE.2013.31.

Sitography

[12] A. Antonielli et al., "*Rapporto 2021 sulla sicurezza ICT in Italia*", 2022 Clusit Report. [Online]. Available on: https://www.mmn.it/wp-content/uploads/2021/05/Rapporto-Clusit_03-2021-web.pdf

[13] Great Learning Team Blog, "*Types of Neural Networks and Definition of Neural Network*", September 28, 2022. [Online]. Available on: https://www.mygreatlearning.com/blog/types-of-neural-networks/

[15] J. Brownlee, "*One-Hot Encode Data in Machine Learning*", Data Preparation, July 28, 2017. [Online]. Available on: https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/

[16] E. Zvornicanin, "*Differences Between Bidirectional and Unidirectional LSTM*", February 5, 2022, [Online]. Available on: https://www.baeldung.com/cs/bidirectional-vs-unidirectional-lstm

[17] Great Learning Team, "*Random forest Algorithm in Machine learning: An Overview*", October 25, 2022, [Online]. Available on: https://www.mygreatlearning.com/blog/random-forest-algorithm

[18] Oden Technologies, "*What is Model Training*", Definition of machine learning training model, [Online]. Available on: https://oden.io/glossary/model-training/

[19] J. R. Choudhary, "*What is Model Validation*", Analytics Vidhya, July 15, 2020, [Online]. Available on: https://medium.com/analytics-vidhya/what-is-model-validation-257686d0253e

[20] A. Mishra, "*Metrics to Evaluate your Machine Learning Algorithm*", February 24, 2018, [Online]. Available on: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234

[21] T. Wood, "*Precision and Recall*", November 202, [Online]. Available on: https://deepai.org/machine-learning-glossary-and-terms/precision-and-recall