



TECHNISCHE UNIVERSITÄT WIEN

TU WIEN

MASTER THESIS

**Comparison of Feature Sets for Detecting Attacks
in Network Traffic**

Author:

Fares MEGHDOURI

Supervisors:

Univ. Prof. Dr.-Ing. Tanja ZSEBY

Dr.techn. Félix IGLESIAS

*A thesis submitted in fulfillment of the requirements
for the degree of Diplom-Ingenieur*

in the

**Communication Networks Group
Institute of Telecommunications**

July 19, 2018

Declaration of Authorship

Hiermit erkläre ich, dass die vorliegende Arbeit gemäß dem Code of Conduct, insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Datum

Unterschrift

Name

“Education is not the learning of facts, but the training of the mind to think.”

Albert Einstein

TECHNISCHE UNIVERSITÄT WIEN

Abstract

Faculty of Electrical Engineering and Information Technology
Institute of Telecommunications

Diplom-Ingenieur

Comparison of Feature Sets for Detecting Attacks in Network Traffic

by Fares MEGHDOURI

The growing amount of encrypted traffic in today's networks makes deep packet inspection infeasible. In addition, high data rates increase the demand for fast processing of network traffic. Attack detection methods need to be based on light feature vectors that can be generated from encrypted network traffic and are easy to extract, process and analyze. So far experts have selected features based on their intuition and previous research works, but there is no general agreement about the features to use for attack detection in a broad scope.

In this work we studied five lightweight feature sets recently proposed in the scientific literature. We compared and evaluated the selected vectors with supervised classification schemes.

Acknowledgements

This work would not have been possible without the help and the guidance of my mentor, Dr. Félix Iglesias Vázquez, his advice and experience were always present while working on this thesis. I would also like to thank Prof. Tanja Zseby, as my teacher and supervisor, she taught me a lot during my studies and was always there when I needed advice.

A special thanks to Prof. Gerald Matz and Prof. Christoph Mecklenbräuer for being amazing teachers, from who I have learned more than just scientific lessons.

Special thanks also go to the colleagues with whom I had the pleasure to study and work, with: Daniel, Valon and Branko.

I would like to thank my family for all the support and encouragement, my mother, my father and my idol, my brother Walid and my uncle, Lahcen.

Contents

Declaration of Authorship	iii
Abstract	vii
Acknowledgements	ix
1 Introduction	1
1.1 Background	1
1.2 Motivation	2
1.3 Goals	2
1.4 Arrangement of The Thesis	3
2 Background Knowledge	5
2.1 Network Traffic Analysis	5
2.1.1 Intrusion Detection Systems	5
2.2 Types of Classification	8
2.2.1 Port-based	8
2.2.2 Deep Packet Inspection	8
2.2.3 Flow-based	9
2.3 Flow-based Analysis	9
2.4 Supervised Classification Algorithms	11
2.4.1 Overview	12
2.4.2 Decision Trees and Random Forests	12
2.4.3 k-Nearest Neighbors	15
2.4.4 Support Vector Machine	17
2.4.5 Naive Bayes	19
2.4.6 Logistic Regression	20

3	Methodology & Experiments	23
3.1	Feature Sets	25
3.1.1	Time Activity vector	26
3.1.2	Consensus vector	27
3.1.3	AGM vector	28
3.1.4	CAIA vector	29
3.1.5	UNSW Argus/Bro vector	31
3.2	Extraction Ability for Encrypted Traffic	33
3.2.1	TLS	34
3.2.2	IPsec	34
3.3	Preprocessing	35
3.3.1	Feature Sets Construction	35
3.3.2	Unidirectional to Bidirectional Flows & Aggregation	36
3.3.3	Labeling	38
3.3.4	Nominal Features	39
3.3.5	Scaling & Log-transformation	42
3.3.6	Features Selection & Reduction	44
3.3.7	Features Importance via Decision Trees	44
3.3.8	Principal Component Analysis	45
3.4	Supervised Analysis	46
3.4.1	Training-Testing.	46
3.4.2	Parameters Tuning	47
3.5	Evaluation	48
3.5.1	Metrics	48
3.5.2	Over-fitting Problem & 5-fold Cross-validation	50
4	Results & Discussion	53
4.1	Extraction Costs Performance	53
4.1.1	Time costs	53
4.2	Training-Testing Classification Performance	57
4.2.1	Time Activity vector	57
4.2.2	Consensus vector	58

4.2.3	AGM vector	59
4.2.4	CAIA vector	60
4.2.5	UNSW Argus/Bro vector	61
4.3	Global comparison	62
4.4	Features importance	63
4.4.1	Time Activity vector	63
4.4.2	Consensus vector	64
4.4.3	AGM vector	64
4.4.4	CAIA vector	65
4.4.5	UNSW Argus/Bro vector	65
5	Conclusions	67
5.1	Conclusions	67
5.2	Submitted Publication	68
5.3	Further Work	69
A	Extraction Tool : Flow Extractor	71
A.1	Structure	74
A.2	Performance	76
B	The NUSW-NB15 Dataset.	79
B.1	Statistics	79
	Bibliography	81

List of Figures

2.1	IDS implementation types	7
2.2	An example showing two hosts communicating. "Flow 1" represent a random flow with two numerical features.	10
2.3	An example of 8 flows with their 2D plot. Two clusters are clearly shown.	11
2.4	An example of an artificial binary data with their respective DT.	14
2.5	The geometric interpretation of kNN classification with k=3.	16
2.6	2D example of a SVM data separation (ref : en.wikipedia.org/wiki/Support_vector_machine).	18
2.7	The logistic function (ref : machinelearningmastery.com/logistic-regression-for-machine-learning/).	21
3.1	Scheme of the conducted experiment [12].	23
3.2	An example of mapping two unidirectional flows into one bidirectional flow.	37
3.3	Nominal to numerical feature mapping example.	40
3.4	The structure of a confusion matrix.	48
3.5	How a data set is divided in order to perform a 5-fold cross-validation.	50
4.1	Time cost of each set. The UNSW set performance is only an estimation and it can differ.	55
4.2	The importance scale used to classify the features. 0 importance is classified as negligible.	64
A.1	A simplified structure of the Flow Exporter.	75

List of Tables

2.1	An example of 8 artificial flows	11
2.2	An example of an artificial binary data with their respective DT.	14
3.1	Features of the Time Activity set.	26
3.2	Features of the Consensus set.	27
3.3	Features of the AGM set.	29
3.4	Features of the CAIA set.	30
3.5	Features of the UNSW Bro/Argus set.	32
3.6	Features of the UNSW Bro/Argus set (continuous).	33
3.7	Traffic encryption and feature sets compatibility.	35
3.8	Nominal to numerical feature mapping example.	40
4.1	Time cost in numbers.	54
4.2	TA classification results.	57
4.3	The optimal parameters for the TA classifiers.	58
4.4	Consensus classification results.	59
4.5	The optimal parameters for the Consensus classifiers.	59
4.6	AGM results.	60
4.7	AGM classifiers optimal parameters.	60
4.8	CAIA classification results.	61
4.9	The optimal parameters for the CAIA classifiers.	61
4.10	UNSW Bro/Argus classification results.	62
4.11	The optimal parameters for the UNSW Bro/Argus classifiers.	62
B.1	Normal to Abnomal distribution of flows.	79
B.2	Attacks distribution.	79

List of Abbreviations

IDS	Intrusion Detection System
ML	Machine Learning
IPS	Intrusion Prevention System
IP	Internet Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
HTTP	HyperText Transfer Protocol
FTP	File Transfer Protocol
DPI	Deep Packet Inspection
TLS	Transport Layer Security
IPsec	Internet Protocol security
DOS	Denial Of Service
ICMP	Internet Control Message Protocol
OSI	Open Systems Interconnection
JSON	JavaScript Object Notation
GT	Ground Truth
TA	Time Activity
CAIA	Center for Advanced Internet Architectures
AGM	AGgregation & Mode
UNSW	University of New South Wales
DT	Decision Tree
RF	Random Forest
NB	Naive Bayes
kNN	k-Nearest Neighbors
LR	Logistic Regression
SVM	Support Vector Machine

PCA	Principal Component Analysis
IANA	Internet Assigned Numbers Authority
IPFIX	IP Flow Information Export
OOP	Object Oriented Programming

Dedicated To My Beloved Parents...

Chapter 1

Introduction

In this chapter, we discuss the reasons that motivated this work, the results that we aimed to and the need for such results in the field of anomaly detection in network traffic.

1.1 Background

Internet traffic classification played a big role in the research related to anomaly detection and prevention. There exist multiple proposals to build systems capable of detecting malware or malicious activity on the network based on different approaches.

One of the basic approaches in order to monitor Internet traffic is keeping track of packets properties. Using statistical features to represent the traffic is also a well known and easy approach for further analysis of the traffic but to the best of our knowledge a few research studied the effect of which feature set to use for this purpose. Most authors use feature sets by default straightforwardly or build their own and keep using it in their research. Studies were conducted focusing on the influence of classification algorithms on the classification results, many of them proposed new classification schemes and/or new representation of the data.

1.2 Motivation

The incredible growth of the Internet nowadays implies that more robust and extremely fast anomaly detection systems need to be developed to secure the network. These systems need first to have the ability to deal with large amount of traffic, second, to raise correct alarms (low false positives and false negatives) and finally they must be able to cope with encrypted traffic since encryption is being progressively adopted in all modern Internet communications.

A considerable part of the research related to anomaly detection and traffic classification does not properly discuss the importance of the feature selection. Authors acritically apply the same features as used in previous works, use default sets, or resort to their own intuition.

In this work, we raise many questions, for instance, *which feature set is most suited for traffic classification? what are the features that carry more information about the behavior of a network?* and finally, *does the choice of the feature set influence the detection results?* Beyond these findings, a main contribution of this work is settling a basis for the discussion of the feature selection problem in network traffic analysis and network security. This issue has been previously addressed in [6] where the authors tried to find the impact of each feature on the classification of traffic and [5] where a meta-study was carried out to rank the most used features in the latest research related to anomaly detection and traffic classification.

1.3 Goals

The need for fast and robust systems brings the discussion to the main element in traffic classification : the features that are used. This work has mainly four goals, we summarized them as follows :

- Comparing lightweight feature sets that have been proposed in the related literature for the last years in terms of classification performance and processing speed. By analyzing these sets a better representation

of the data and a better classification performance can be achieved in the future.

- We also discussed the importance of each feature in each studied set, which allowed us to have a general idea on the most relevant features in an IDS (Intrusion Detection system).
- From the previous results, we wanted to propose enhancements on the previously used sets.
- As a last step, we analyzed the relationships between features and attack families, exploring which features were relevant to identify specific types of network attacks.

1.4 Arrangement of The Thesis

This thesis is divided into five chapters :

- In chapter 2 we introduce background knowledge related to network traffic analysis and machine learning classification algorithms.
- In chapter 3 we introduce our sets, the methodology used and the evaluation of the results.
- In chapter 4 we discuss performance results for both extraction time costs and classification performance, we then propose enhancements to the previous sets.
- Finally, we enclose this work with a conclusion in Chapter 5 regarding findings and research that can use our work as a baseline.

Appendices A and B introduce the tool used in order to extract different sets and the dataset used in this work.

Chapter 2

Background Knowledge

In this chapter two topics are mainly discussed. First, we explain what is meant by network traffic analysis and the fundamental ideas behind. Second, we introduce six ML (Machine Learning) algorithms in the scope of supervised classification.

2.1 Network Traffic Analysis

In order to detect anomalies in Internet traffic we need well defined methodologies. Performing analysis on network traffic is a way to have a general idea on the state of a network. Previous research presented many approaches serving the same goal : detecting anomalies accurately. In this section we present the basic structures of these systems.

2.1.1 Intrusion Detection Systems

An IDS (Intrusion Detection System) is defined as any system placed in a network in order to notify the administrator about any suspicious activity or an anomaly which changes the normal behavior of this network. This concept was first introduced in 1980 by James Anderson [2]. The main task of these systems is to raise an alert when necessary, but more advanced systems (IPS - Intrusion Prevention Systems) can additionally react by their own and execute preloaded instructions, for instance : *filtering*. Tunning these systems to enhance their detection accuracy (less false positives and negatives) is still a

challenge. Moreover, these systems need to be robust, able to learn new patterns and adapt to the evolving network characteristics (dynamic reaction).

In this work, a light IDS is used to monitor Internet traffic . It can be placed everywhere in the network depending on the resources. We present the overall structure of such systems and the different types that have been developed in the past. Mainly two categories exist:

Network intrusion detection systems. This type of implementation is the simplest. The IDS is installed in a defined position on the network (usually near to a firewall). If this position is properly chosen, the system will be able to monitor all the traffic going in and out of the network. An alarm is announced only if the current behavior of the network mismatches the usual behavior. However, scanning all traffic passing through a single point is a bad idea (it will create bottlenecks) and requires a lot of computational efforts. This can be solved by reducing the number of operations needed, packets filtering/selection or even using a multi-stage IDS. Figure 2.1a shows a possible implementation.

Host intrusion detection systems. The second type is a host based IDS. The system here is installed in a network host and keeps track only of the traffic going in and out of the specific host. The administrator is then notified whenever a malicious activity is detected affecting the controlled hosts. Advanced host based IDS can even prevent the spread of malicious activities to other hosts. An example of such system is an *anti-virus* software installed in hosts. Figure 2.1b shows the respective implementation.

After discussing the two main IDS structures, we now discuss two main detection methods recently proposed. They are explained in what follows :

Signature-based. This type of detection is based on behavior matching, in other words, the system has a database preloaded with attack patterns. The system keeps comparing the current measurements from the network with

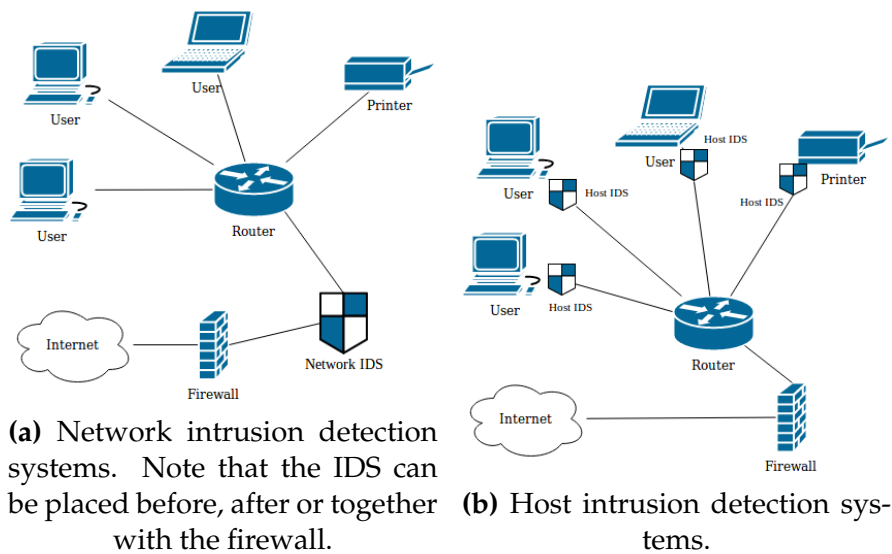


Figure 2.1: IDS implementation types

all entries in the database. It raises an alarm only if the network traffic pattern matches one of the attack patterns. A pattern is defined as a set of features having a certain value, e.g, the packet load signature used for worms detection (in the case of non-encrypted traffic). Well known applications of such detection technique are anti-viruses where databases are updated regularly. This detection method is really effective with pre-known attacks hence the false positive rate is really low. Nowadays, the main issue is reacting to new attacks (or even old attacks with a different signature), where the system is obviously not capable of fully adapting itself to changes.

Anomaly-based. This type of detection is suited for applications where anomalies or attacks are not known or change dynamically. Most of the systems of this type are based on mathematical formulations with parameters that can be adjusted. An alarm is raised whenever a statistical condition is met or network traffic measurements are either out of range or exceeds pre-defined thresholds. The output of such systems is usually binary (*normal* or *abnormal*). The main drawback of this approach has to do with high false positive rate, however, many parameters tuning methods were developed to overcome this issue and even an adaptive tuning based on pattern learning can be used along with the detection module.

One should also be aware that there exists no perfect IDS in the sense that its accuracy is 100% nevertheless, we can make use of ML technique to both : construct a robust system and keep tuning it via dynamic learning processes.

2.2 Types of Classification

Signature based or anomaly based, an IDS needs anyway to gather measurements from the network. The cost, the possibility of extraction and the data extracted for each measurement type are not the same. We explain three classification schemes based on different features.

2.2.1 Port-based

It is one of the simplest ways to classify both TCP and UDP traffic. This is due to the fact that many services are already known and associated with famous port numbers (HTTP:80, FTP:20,21...). The classifier needs only to parse the TCP/UDP header and then classify traffic based on a prestored database (provided for instance by IANA [9]). Despite the easiness and the low resource consumption needed, this method is not reliable. An attacker can easily break the detection system for the only reason that not all port numbers are well and preassigned; even worst, tunneling over known port numbers can be used and an attacker can hide a malicious activity pretending to be a normal service therefore.

2.2.2 Deep Packet Inspection

DPI (Deep Packet Inspection) is the most difficult method to achieve. With DPI we break one of the most important rules in the Internet : privacy. The idea behind is analyzing the content of packet payloads. By doing so, many complex attacks can be detected. Many works propose techniques to do it, one of the intuitive methods is using well known signatures and try to match

them with the payload. The encryption of traffic is a big problem for this approach. In the case of TLS or IPsec usage, the payload is not available to a third party and therefore, no useful information can be extracted. Another issue is the bad support and the lack of signatures publicly available from well known service providers. If a service provider does not want to make his protocol structure available to the public (open-source), it would be impossible to recognize its traffic (e.g., Skype).

2.2.3 Flow-based

Flow statistics can be used to classify network traffic as well. A flow is a group of packets that belongs to a same connection between two hosts in a defined time window. It is represented by a key and a set of features extracted from traffic measurements. That is why it is a good approach to represent network traffic. Classification schemes based on flow statistics need only header fields, which makes it faster than DPI and offers more information than the port-based method. An endless amount of network traffic features are available and can be used but a few of them really carry meaningful information, when considering the task of network traffic classification. However, using a high number of features causes challenges in further steps (e.g., curse of dimensionality [10]), therefore, it is recommended to choose the features wisely.

2.3 Flow-based Analysis

All vectors studied in this work belong to the flow-based type. In this section, we discuss in detail this type of representation.

A network traffic flow is defined as the set of packets exchanged between two hosts, a host and a group of hosts (multi-cast) or a host and all hosts in a network (broadcast). A flow can be either unidirectional (data transfer in one direction) or bidirectional (data transfer in both directions). We extend this definition further by setting a key to each flow so that we can distinguish it

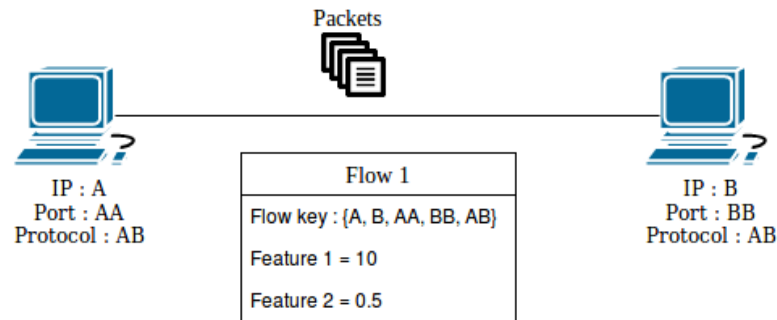


Figure 2.2: An example showing two hosts communicating. "Flow 1" represent a random flow with two numerical features.

from other flows. An example of a flow key is the well known 5-tuple that contains: the source IP address, the destination IP address, the source Port, the destination Port and the protocol $\{sIP, dIP, sPort, dPort, Protocol\}$. In order to reduce the computational resources, we define two parameters to limit the flow duration (otherwise, the IDS needs to keep the connection open forever). The first one is the *timeout*, which is the time needed before the flow is set to "finished" in case where no activity is detected. Second, *active time*, which is the maximum flow duration even if an activity is present. This is mainly needed in the case of long continuous flows, which are preferably divided into smaller sub-flows to maximize the information gain (many flows provide more detailed information than a single large flow).

The set of features constructing each flow are parsed simply from the group of packets belonging to that flow. Sometimes, mathematical operation or statistical transformation are also used to map a list of numerical values into only one (the mean for instance). Figure 2.2 shows an example of two hosts exchanging packets and the extracted flow.

Now back to the main topic, how can a flow be used to classify Internet network traffic? Geometrically speaking, each flow represents a point in an n -dimensional space where n is the number of flow features. By plotting all these points we see clusters where each one should represent a class or a category with the same behavior in the respective network traffic. To give an example, let us consider some artificial flows with two features each and plot the respective 2D space (Figure 2.3). Table 2.1 shows 8 flows with

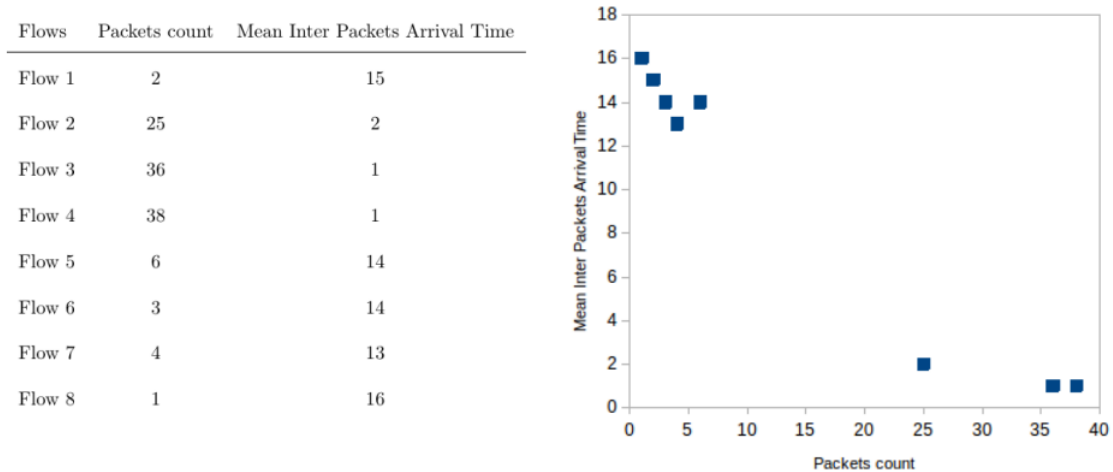


Figure 2.3 & Table 2.1: An example of 8 flows with their 2D plot. Two clusters are clearly shown.

two features, the first one is the number of packets transmitted between the hosts, the second feature is the mean inter arrival time (it is the mean time between received packets). Note that this is an extremely simplified scenario of a DOS (Denial Of Service) attack, the attacker tries to send a massive amount of packets in a short time period which will consume the victim's resources quickly and thus the victim's host will fail. From Figure 2.3, we notice that two clusters are formed, the left cluster consists of flows with a low number of packets and a large mean inter-arrival time which is considered here as a normal behavior; the right cluster consists of flows with a large number of packets and a small mean inter-arrival time. A reasonable system will therefore detect and differentiate between the two clusters and more advanced systems can even predict that this is a DOS attack.

2.4 Supervised Classification Algorithms

In this section, six of the most known ML classification algorithms that were used in this work are briefly explained.

2.4.1 Overview

This work uses supervised classification for analysis tasks. Supervised classification means that the data analyzed by ML algorithms is already labeled (the label of each flow is known). A comparison of the predicted labels and the original ones allows us to measure the performance of correctly classified flows being able to benchmark both : the features sets and the ML algorithms.

The main idea behind most ML algorithms is similar : create a model that best fits the mapping data \rightarrow label by learning from *training data* with labels. Once the model is ready, we test it with *test data* and we get predicted *test labels* based on the model (note however that one of the algorithms used in this work is not a model based algorithm –kNN–, it is a lazy learning algorithm and will be explained later). By comparing the predicted labels and the original ones using predefined metrics (explained in 3.5.1), we can study how good is our model.

2.4.2 Decision Trees and Random Forests

Decision Trees One of the most and widely used algorithms in a supervised classification is the well known DT (Decision Trees) algorithm. Since their invention in 1984 by Charles J. Stone [16], it received a lot of interest in the ML community, where a lot of work has been done to improve their concept and the way they are implemented. Originally, DT were not only used in ML but also in data mining and statistics. One of the examples of their usage is the prediction of a house price by observing related collected data.

We used DT in this work simply to classify Internet network traffic by building a model capable of predicting the class of a flow using its features.

The main idea behind the algorithm is the following : a tree structure is used, which is based on conditional leaves. We start by defining defining data points as following :

$$D^i = \{d_1^i, d_2^i, \dots, d_n^i, y^i\}$$

where d_j^i for $j = 0, \dots, n$ are the features values and y^i is the targeted category. Next, consider a single data point $D^1 = \{d_1^1, d_2^1 \dots d_n^1, y^1\}$, we then create our tree by starting from a root node and continue further by extending leaves from this main node and mark the path based on conditions made on each of the data point's feature values (d_j^1) at each node. At the end we label the whole respective path as y^1 . Once the graph is finished, we end up with a number of leaves that represent the classes (note that multiple different leaves can represent the same class). Later on, when a new data point arrives (D^2), the search algorithm will try therefore to find the suited path using the features values $\{d_1^2, d_2^2 \dots d_n^2\}$. Once we reach the last leaf, the class is then well defined. The next example helps understanding DT better. Table 2.2 shows a simple binary data and Figure 2.4 shows its respective decision tree.

We assumed the root node to be A. We checked the values of this feature, then we concluded that two leaves should be derived from it, we had a 0 leaf and a 1 leaf. We created a list on each leaf that summarized which classes are present with the main condition $A = 1$ or $A = 2$. Two cases were present here : first, if a one to one mapping between the feature value and the class existed, then we set the class to its respective one (for instance $A = 1$ means that the category is Blue). Second, if there is no simple mapping, the leaf should be further divided (the case $A = 0$) and the whole algorithm is repeated again. Let us continue with the second feature B, since it also took binary values, two leaves were created from it. We repeat the previous steps and we end up with two cases: for $B = 0$ we have a single one to one mapping that gave Red. For $B = 1$ we are stuck again so we have to extend further. After many iterations, the graph in Figure 2.4 was created. Later, any new point that arrives can be then classified as one of the three classes only by finding its path within the tree.

We would like to point out to a few things here :

- The order of features choice in reality is not arbitrary, a few metrics can be used (we cite : Gini Impurity, Information Gain and Variance Reduction). These metrics ensure that "the right" feature is chosen in

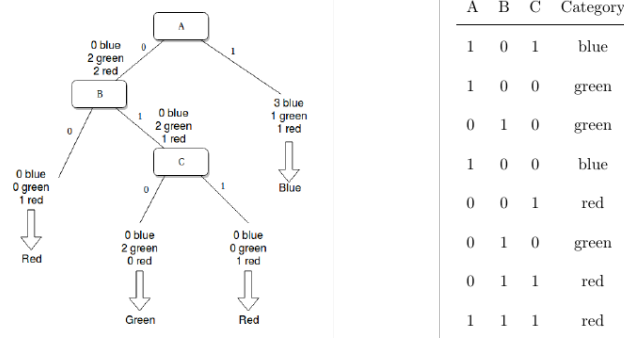


Figure 2.4 & Table 2.2: An example of an artificial binary data with their respective DT.

the next iteration depending on the amount of information gain.

- Many algorithms were developed to execute the explained procedure in an optimized manner. For example : ID3 (Iterative Dichotomiser 3) that uses the Information Gain for the choice of the next feature, C4.5 that is based on ID3 but flexible to work with continuous data, C5.0 that is optimized to use less memory and rule-sets for splitting and CART (Classification and Regression Trees) that is similar to C4.5 but can support numerical values as target class (a regression).
- The complexity of a balanced tree at each node is $O(n_{features} n_{samples} \log(n_{samples}))$ which implies for the whole tree a complexity of $O(n_{features} n_{samples}^2 \log(n_{samples}))$. This is not always true but only if the tree is balanced. However, the complexity can be reduced by keeping state of the distribution function of the classes at each node (this is the implementation used by Scikit-learn [15]), which gives a final complexity of $O(n_{features} n_{samples} \log(n_{samples}))$.
- To overcome the over-fitting problem (low bias, but very high variance), two simple solutions exists. First, by setting the maximum depth of the tree which will prevent the algorithm from creating leaves for a very sparse features distribution and group such data points in a main branch. Second, we can simply remove leaves with a few samples because most probably they represent noise.

Random Forests This classification method is based on DT and is used to improve its performance. Basically by using multiple DT trained with different portions of the training data each tree will have its own structure. When a new data point comes, it will be classified by all trees and the RF (Random Forests) will come into play and takes the majority vote of the predicted class.

Random forests help us to reduce the variance of the prediction which is also a solution for the overfitting problem. The new classification model performs much better however, the computational power is increased linearly with the number of trees and the user loses the easy interpretability of the classification criteria (in DT, it was quite obvious).

2.4.3 k-Nearest Neighbors

One of the top ML algorithms used for both classification and regression is the kNN (k-Nearest Neighbors). It was first introduced in 1952 by Fix & Hodges in an unpublished paper since then, many changes and enhancements were introduced to make it more robust and faster. The state of the art of this algorithm is that it needs no training procedure (at least comparing to other algorithms). This is what is known nowadays as Lazy Learning.

The idea is the following : suppose that we have a dataset that contains data points of such as :

$$D^i = \{d_1^i, d_2^i, \dots, d_n^i, y^i\}$$

where d_j^i for $j = 0, \dots, n$ are the features values and y^i is the original class to be predicted. The algorithm set all the training points in an n -dimensional space and label each one with y^i . Next, when a testing point comes, the algorithm calculates a specific distance (Euclidean for instance) between the new data point and all existing ones, then sort them incrementally. Next, the algorithm takes the first k distances and perform a majority vote on the label of these points which will then give us the predicted class. Figure 2.5 shows an example where $k = 3$, there we can see 8 training points each containing two features in a 2D space. The points are binary classified (left cluster =

0, right cluster = 1). When a new testing point arrives (in the middle), the algorithm with $k = 3$ will then try to calculate the Euclidean distance for example to all 8 points and keep only the three shortest. We can see that among these three, two belongs to the "right" class and one to the "left" class. Any reasonable majority vote will then predict the "right class" (or a 1) for this new data point.

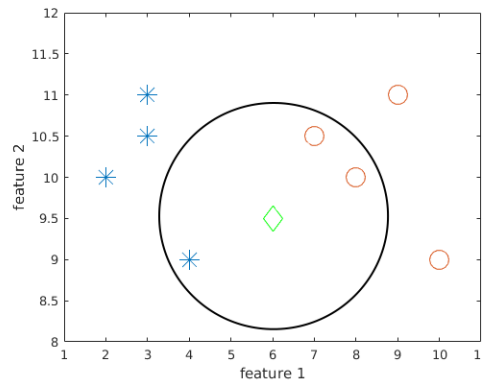


Figure 2.5: The geometric interpretation of kNN classification with $k=3$.

Mathematically speaking, the formulation is :

$$\hat{y} = \arg \min_y \sum_{\Omega} I(y = y^i)$$

where \hat{y} is the predicted category, Ω is the set of "k" nearest neighbors and y^i is the category of each of the nearest neighbors.

We point out to a few things here also :

- As said before, the first step is to calculate distances to all other points. Many distances can be used and research were conducted and studied the impact of choosing each distance. Some of the famous distance metrics are : the Euclidean Distance, Chebychev Distance, Manhattan Distance... In our context, we simply use the Euclidean Distance which is based on the l_2 norm.
- Another issue is the choice of the parameter k . It is true that the choice is random but we need to clarify some mistakes that can be made. For

instance, choosing an even number will cause a confusion in the case where the classes are equal (50% in one class and 50% in another one in a binary classification for instance) and, therefore, the majority vote can not be executed, that is why an odd value need to be chosen. Another issue is choosing a large value, besides the heavier calculations need to be done, choosing a large value will cause a smoother boundaries between classification regions, hence a miss-classification (bias) can occur. So what is the best value? Well, there is no best value for k , but there is a heuristic method to find the most suited value for a certain classification. It is sufficient to run a loop that measures the classification error for different values of odd k . The error will not converge when incrementing k but a minima should exists somewhere (most probably at low values), this minima can be chosen for further analysis.

- kNN is also known to have difficulties when dealing with high dimensional data, the reason behind is the computation of distances. Basically, if the dimension is too large, the distances will converge to the same value (vectors will be equidistant) and thus the algorithm will have hard time to find the nearest neighbors to a given point. Features selection/reduction techniques are recommendable before applying kNN.
- A different flavor of this algorithm is the Weighted Nearest Neighbors where a set of points have weight and thus, contribute differently to the vote. This is often useful when the data is coming from sources with different credibility.

2.4.4 Support Vector Machine

Another ML algorithm used for supervised classification and regression is the SVM (Support Vector Machine). First invented by Vladimir N. Vapnik and Alexey Ya. Chervonenkis in 1963. Later, it received a lot of attention and

researchers introduced many enhancements e.g., the notion of kernels which is used to maximize the margin between hyperplanes.

The basic idea behind SVM is the following : building a model that finds gaps between training data points in an n -dimensional space and consider them as borders. Later, for each data point in the test set, it finds on which side of the border the point lays.

Figure 2.6 helps us understand SVM better. We see a group of points in a 2D space and three hyperplanes (lines in this case). The lines have different rotations and try to find the largest gap between the two clusters. This operation is also known as "margin maximization". H_3 is clearly here the best line for separating the clusters. When a new point comes, the label is predicted by looking at which side the sample belongs.

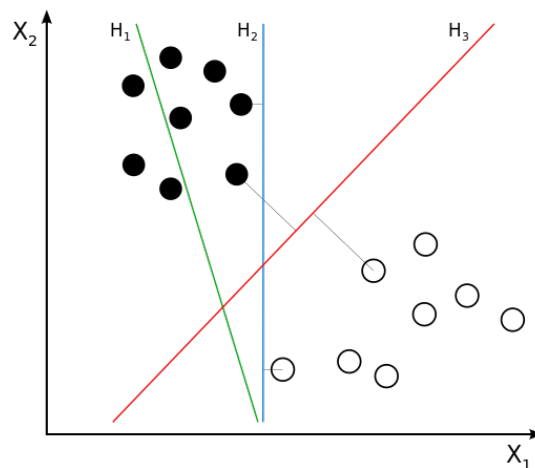


Figure 2.6: 2D example of a SVM data separation (ref : en.wikipedia.org/wiki/Support_vector_machine).

Sometimes, the data is not linearly separable and constructing hyperplanes is not possible. In order to solve this problem, the n -dimensional space can be mapped into a higher dimension space where the data is linearly separable. To keep calculations light, one uses a method called kernel functions. Two modes of SVM exists, linear kernel SVM and non linear kernel SVM:

Linear SVM. Is used in the original space and has two variants : Hard margin; where the data is linearly separable, and soft margin, where the data is not linearly separable and some error is allowed. Note that the idea behind

linear SVM can be explained with the 2D example. In a 2D space, assume that we have two clusters and a margin between them. Try to find a point from the extreme of each cluster and create two parallel hyperplanes where each crosses one of the points. Next, try to adjust the rotation of the hyperplanes until maximizing the distance between them. Finally, create a main hyperplane which lays in the middle and is parallel to the previous ones.

Nonlinear SVM. Instead of working in the original space, we transform it into a higher dimensional space (the transformation does not have to be linear); then, we perform the same steps. The difference here is that the dot product used in the mathematical formulation in the Linear SVM is transformed into a kernel function.

SVM parameters can be adjusted manually or using a Grid Search (discussed in Section 3.4.2).

SVM works really good with high dimensional data. It is also memory efficient and its kernel functions are diverse.

2.4.5 Naive Bayes

The NB (Naive Bayes) classifier is a pure statistical-probabilistic classifier. The main idea is to apply the Bayes theorem with the assumption that the data features are independent. The targeted class is the result of computing probabilities between features that contribute independently. There is a need for statistically independent features (note that PCA for instance can be applied beforehand to eliminate correlation between features).

Let D^i be a new data point such as :

$$D^i = \{d_1^i, d_2^i, \dots, d_n^i, y^i\}$$

where d_j^i for $j = 0, \dots, n$ are the features values and y^i is the targeted category. Applying the Bayes theorem will lead to :

$$P(y^i | d_1^i, d_2^i, \dots, d_n^i) = \frac{P(y^i)P(d_1^i, d_2^i, \dots, d_n^i | y^i)}{P(d_1^i, d_2^i, \dots, d_n^i)}$$

Since the features are assumed to be independent, the following is valid :

$$P(y^i | d_1^i, d_2^i, \dots, d_n^i) = \frac{P(y^i) \prod_{k=1}^n P(d_k^i | y^i)}{P(d_1^i, d_2^i, \dots, d_n^i)}$$

where the multi-variables conditional probability is replaced with the marginal probabilities product. Then since $P(d_1^i, d_2^i, \dots, d_n^i)$ is a constant, the relation can be simplified to

$$P(y^i | d_1^i, d_2^i, \dots, d_n^i) \propto P(y^i) \prod_{k=1}^n P(d_k^i | y^i)$$

and the estimate of the class is computed as :

$$\hat{y} = \arg \min_y P(y) \prod_{k=1}^n P(d_k^i | y^i)$$

The last formula is the core of all NB classifiers, the class can be then estimated once $P(y)$ and $P(d_k^i | y^i)$ are available (can be computed via a MAP (Maximum A Posteriori) estimator). $P(d_k^i | y^i)$ can be assumed to follow different distributions and therefore, different NB classifiers variants exist. In this work, $P(d_k^i | y^i)$ follows a Bernoulli distribution.

In the training procedure, probabilities are computed and saved in memory. Later, these probabilities and observations together are used to predict classes.

2.4.6 Logistic Regression

LR is also a statistical model used for both classification and regression. It was first developed by David Cox in 1958. Many versions of it exist however, the simplest is the binary classifier (two classes). The idea behind the binary LR is mapping input features values probabilities into output probabilities, which in turns give the predicted class. Similar to Naive Bayes, Logistic Regression assumes statistically independent features. Having said that the output is nothing but probabilities, many researches consider LR not

as a classifier; however, by using a thresholding function, it can be turned into a binary classifier.

Logistic regression is based on a core function called the logistic function. This function takes input values and maps them into probabilities (between 0 and 1). The logistic function has an S shape, Figure 2.7 shows such a function.

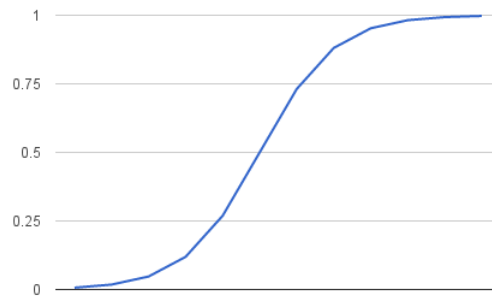


Figure 2.7: The logistic function (ref : machinelearningmastery.com/logistic-regression-for-machine-learning/).

Starting from conditional probabilities, class probabilities can be computed. Later, using the logistic function, the class can be predicted. Converting these probabilities needs coefficients, which will be adjusted during the training phase and used later in the testing phase.

LR has basically three parameters to tune. We are only interested in the "penalty parameter" (the other two parameters are adjusted automatically using parameter tuning, discussed in Section 3.4.2). The penalty represents the norm used in the penalization, this is an approach used in the training stage to avoid over-fitting, it is also called regularization. In this work the L_2 norm was used as penalty.

We point out here that the LR classifier is similar to NB meaning that the classification results should be similar. more about this is discussed in Chapter 4.

Chapter 3

Methodology & Experiments

In this chapter, we introduce the analysis framework structure, the five feature sets and explain afterwards how the experiments were conducted. Precisely, we explain how we constructed every set, present the different data transformations performed such as nominal transformation and features selection/reduction etc. Finally, we give details about the supervised analysis approach and the respective evaluation methodology using well known metrics.

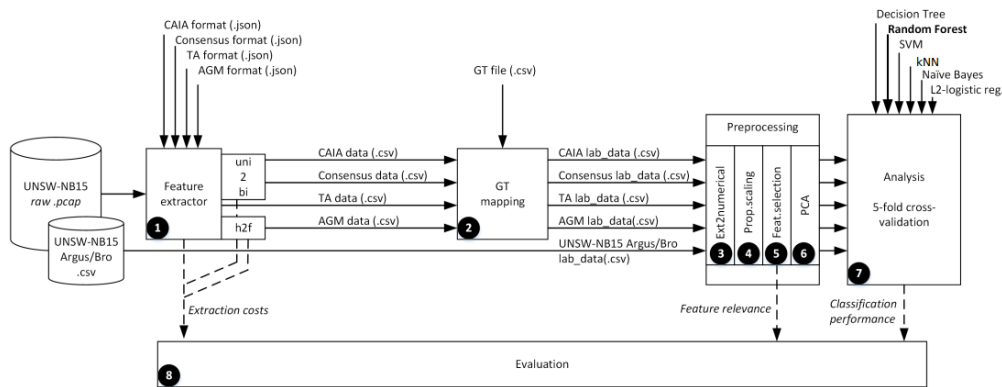


Figure 3.1: Scheme of the conducted experiment [12].

Figure 3.1 shows the framework's structure, it had multiple stages, every stage is explained briefly in what follows.

1. **Feature extractor** : in this stage, we used the Flow Exporter to restructure the dataset as described by the studied sets (the UNSW Argus/Bro was already constructed and included with the original dataset). Section 3.1 presents the five sets, Section 3.3.1 explains how the sets were

constructed and Appendix A shows more details about the FLOW Exporter tool.

2. **GT mapping** : labeling the data was the second step that we carried out. Each flow was given a label (normal (0) or malicious (1)) using a Ground Truth file included with the dataset, simply if a match was found, a column was then added (called Label). However, the AGM format needed a different type of labeling. More details are discussed in Section 3.3.3.
3. **Nominal to numerical mapping** : next step was getting rid of nominal features and preparing the data for the classification algorithms. Many ML algorithms deal only with numerical inputs and thus, we decided to transform all sets into only numerical data for the sake of uniformity. Dummy variables were used, which lead into dimensionality increase. However, some of the nominal features were not relevant, which implied that the dimensionality could be reduced again in further steps. More details are discussed in Section 3.3.4.
4. **Scaling** : scaling data before feeding it to a ML algorithm is another preprocessing step that many researchers ignored in the past in spite of being necessary. Using unscaled data might lead to a bias in the classification notably with algorithms that use distance as a basis for classification decision. Further details about scaling are discussed in Section 3.3.5.
5. **Feature selection** : in order to reduce dataset dimensionality some methods have been proposed in the literature. Feature selection covers a set of techniques that aim to keep only the features that are meaningful for the analysis. In this work, we used DT feature importance as a rule in order to keep only relevant features (relevancy $\neq 0$). More about this topic is discussed in Sections 3.3.6 and 3.3.7.
6. **Principal component analysis** : PCA is a reputable method used to find and remove feature correlations in datasets. On the other side, network

traffic features are well known to show high correlation. So applying PCA is a solution to reduce problems related to feature dependencies during the analysis. we used PCA first to help the classification algorithms perform better (Naive Bayes is known to perform poorly when fed with correlated data). In Section 3.3.6 and 3.3.8 we give more details.

7. **Analysis and Cross-validation** : after the preprocessing, ML algorithms performed supervised classification on the preprocessed datasets. Additionally, a 5-fold cross-validation was included in order to make sure that the results are valid and performance are stable. Sections 3.4 and 4.5 discuss this in depth.
8. **Evaluation** : the core of this work was the evaluation and comparison of performance results for each set. Additionally to the preprocessing times, we used the predicted labels from the supervised classification to evaluate the classification performance. Furthermore, we carried out some statistical analysis to evaluate the relevancy of each feature in identifying a certain type of attacks. This step is discussed in depth in Chapter 4.

An intermediate step was introduced between steps 1 and 2. At the output of the Flow Exporter, all sets were extracted as a group of unidirectional flows. Afterwards, CAIA and Concensus were converted into bidirectional flows. The AGM set was on the other hand aggregated by source. More details about this topic are discussed in Section 3.3.2.

3.1 Feature Sets

This section introduces the five studied feature sets, the way they were constructed and the adjustments that we had to perform for a proper analysis.

3.1.1 Time Activity vector

The time activity vector was first presented in [8]. It represents the time behavior of a network flow. Using a fixed window size, this vector gathers statistics on the behavior of the information exchange between two hosts in a unidirectional manner. Table 3.1 shows the features contained in the TA set and their respective description. The set contains 15 features plus 5 key features (some key features are also considered as vector features). The key here is the well known 5-tuple: $\{sourceIPaddress, destinationIPaddress, sourceport, destinationport, protocol\}$. Note that in the experiments, the source and destination IP addresses were removed because they were relevant only in the identification of the flows and mapping them into dummy variables will cause a massive increase in dimensionality.

Key			
srcIP	source IP address	pkts_per_seconds-active	average packets per active-second transmitted
dstIP	destination IP address	maxton	maximum amount of consecutive seconds that the flow showed activity
srcPort	source Port		
dstPort	destination Port		
protocol	Protocol		
Features			
srcPort	source Port	minton	minimum amount of consecutive seconds that the flow showed activity
dstPort	destination Port		
protocol	Protocol		
bytes	total amount of data transmitted	maxtoff	maximum amount of consecutive seconds that the flow did not show activity
pkts	total amount of packets transmitted	mintoff	minimum amount of consecutive seconds that the flow did not show activity
secondsactive	number of seconds when the flow was active	interval	number of activity intervals
bytes_per_seconds-active	average data per active-second transmitted (bytes)		

Table 3.1: Features of the Time Activity set.

The TA set gathers unidirectional flows which allows a better representability of both directions in an exchange. However, few attacks are better classified when the behavior is represented in a bidirectional manner thus, there is

always a trade-off on which mode to choose. Further details about the later are discussed when presenting our results.

3.1.2 Consensus vector

The consensus vector is formed by the most used statistical features in last years research according to [5]. There, the authors carry out a meta-analysis and disclose at the end which features are repeatedly used. It is constructed by 23 features and 5 key features where 3 among them (*sourceport*, *destinationport*, *protocol*) are kept as part of the vector itself. Table 3.2 shows the corresponding features and their description.

Key			
srcIP	source IP address	median_srcPktLength	the median value of packet lengths in forward direction
dstIP	destination IP address	min_srcPktLength	the smallest packet sent in the forward direction
srcPort	source Port	median_srcPktIAT	the median value of inter arrival time in forward direction
dstPort	destination Port	variance_srcPktIAT	the variance of inter arrival time in forward direction
protocol	Protocol	max_dstPktLength	the largest packet sent in the backward direction
Features		mode_dstPktLength	the packet length that is most present in backward direction
srcBytes	amount of data sent in the forward direction	median_dstPktLength	the median value of packet lengths in backward direction
srcPkts	amount of packets sent in the forward direction	min_dstPktLength	the smallest packet length in the backward direction
dstBytes	amount of data sent by the backward direction	median_dstPktIAT	the median value of inter arrival time in backward direction
dstPkts	amount of data sent by the backward direction	variance_dstPktIAT	the variance of inter arrival time in backward direction
srcPort	source Port		
dstPort	destination Port		
protocol	Port		
duration	duration of the flow (in secondes)		
max_srcPktLength	the largest packet length in the forward direction		
mode_srcPktLength	the packet length that is most present in forward direction		

Table 3.2: Features of the Consensus set.

In the original paper [5], the vector contains two features labeled *server – client* and *client – server*. These two features differentiate both directions in a single connection which makes the set bidirectional. Instead of using these two features, we discarded them and mapped each of the original features into two sub-features, the first represented the same feature value in the forward direction and the second represented its value in the backward direction. An example of this mapping is the *PktCount* feature which was mapped into *srcpPktCount* and *dstPktCount*. Additionally, authors propose to use a couple of statistical transformations on many features; in this work, we used the following : $\max(\text{feature})$, $\min(\text{feature})$, $\text{mode}(\text{feature})$, $\text{median}(\text{feature})$, where the mode was the most recurrent value in a list. These transformations were applied to both *PktIAT* and *PktLength*.

3.1.3 AGM vector

The AGgregation and Mode vector was first presented in [7]. It introduces a novel representation based on modeling host behavior by aggregating flows. It can be used to monitor attack sources as well as victims. In this work we focused on attackers (sources) where we tracked the activity of a host sending normal or abnormal traffic. The basic features used by the AGM vector were *destinationIPaddress*, *sourceport*, *destinationport*, *protocol*, *TTL*, *flags*, *length* and *numberofpackets*. As a key only the *sourceIPaddress* was used. Each of these features (excluding *numberofpackets*) was then mapped into three sub-features by using statistical transformations given by : $\#(\text{feature})$: number of distinct values, $M(\text{feature})$: the mode (most recurrent value) and $\#pkts[M(\text{feature})]$: number of packets sent to the statistical mode. That resulted in 22 features which are shown in Table 3.3 with their respective description.

The AGM vector needed an extra step during the extraction, which is the aggregation (a set of flows were grouped together if they belonged to the same host). This extra step might be a deficiency when evaluating the extraction performance. However, we will discuss this in a later section.

Additionally, when extracting the AGM vector, a larger time window was used. This will allow each AGM entry to parse more packets/flows and, thus, the behavior would have been better represented.

Key			
srcIP	source IP address	mode_protocol	the protocol that showed the most recurrence
Features		pkts_mode_protocol	the number of packets sent with the mode protocol
#dstIP	the number of distinct destination IP addresses	#TTL	the number of distinct TTL values
mode_dstIP	the destination IP address that showed the most recurrence	mode_TTL	the TTL value that showed the most recurrence
pkts_mode_dstIP	the number of packets sent to the mode	pkts_mode_TTL	the number of packets sent with TTL mode
#srcPort	the number of distinct source port	#TCPflag	the number of distinct TCP flags
mode_srcPort	the source port that showed the most recurrence	mode_TCPflag	the TCP flags that showed the most recurrence
pkts_mode_srcPort	the number of packets sent to the mode	pkts_mode_TCPflag	the number of packets sent with mode TCP flags
#dstPort	the number of distinct destination port	#pktLength	the number of distinct packet's lengths
mode_dstPort	the destination port that showed the most recurrence	mode_pktLength	the packet length that showed the most recurrence
pkts_mode_dstPort	the number of packets sent to the mode	pkts_mode_pktLength	the number of packets sent with mode packet length
#protocol	the number of distinct protocols	pkts	the number of packets exchanged

Table 3.3: Features of the AGM set.

3.1.4 CAIA vector

CAIA stands for the Center for Advanced Internet Architectures at the Swinburne University of Technology, where a research group used this vector for the first time in [18] to the best of our knowledge. Afterwards, many research papers use the same set, such as [11], [17] and [20]. The set represents a pure

statistical behavioral model of Internet traffic, it uses four statistical transformations to map basic features. Additionally, it considers both directions, therefore, every feature is mapped into 8 sub-features as shown in Table 3.4.

Key			
srcIP	source IP address	min_dstPktLength	the minimum packet's length in backward direction
dstIP	destination IP address	mean_dstPktLength	the mean packet's length in backward direction
srcPort	source Port	max_dstPktLength	the maximum packet's length in backward direction
dstPort	destination Port	stdev_dstPktLength	the standard deviation of packet's length in backward direction
protocol	Protocol	min_srcPktIAT	the minimum inter arrival time in forward direction
Features		mean_srcPktIAT	the mean inter arrival time in forward direction
protocol	protocol	max_srcPktIAT	the maximum inter arrival time in forward direction
duration	the duration of the flow	stdev_srcPktIAT	the standard deviation of the inter arrival time in forward direction
srcPkts	the number of packets transmitted in forward destination	min_dstPktIAT	the minimum inter arrival time in backward direction
srcBytes	the number of bytes transmitted in forward destination	mean_dstPktIAT	the mean inter arrival time in backward direction
dstPkts	the number of packets transmitted in backward destination	max_dstPktIAT	the maximum inter arrival time in backward direction
dstBytes	the number of bytes transmitted in backward destination	stdev_dstPktIAT	the standard deviation of the inter arrival time in backward direction
min_srcPktLength	the minimum packet's length in forward direction		
mean_srcPktLength	the mean packet's length in forward direction		
max_srcPktLength	the maximum packet's length in forward direction		
stdev_srcPktLength	the standard deviation of packet's length in forward direction		

Table 3.4: Features of the CAIA set.

The CAIA is intuitive, simple and purely numerical, so it minimizes the required preprocessing steps. Additionally, four features were added since some works use the same basic set plus the following *TCP* flags: *SYN*, *ACK*,

FIN, *CWR* (note that each flag is mapped into two sub-features representing both directions).

3.1.5 UNSW Argus/Bro vector

The UNSW Argus/Bro feature set is included and described in the dataset used in this work (see Appendix B). In [14] and [13] authors describe the set and perform analysis using it. The set has 48 features, 5 among them are used as a flow key. Most of the features are extracted with the Bro and Argus tools. Tables 3.5 and 3.6 show the features with their respective description. The features extraction in this set is not always possible (requires deep inspection and other techniques that are not explained in the original papers) and thus, makes the set unusable in some cases (encrypted traffic, for instance). Even when dealing with unencrypted traffic, extracting some of the features is complicated (keeping state of previous connections) and will lead into a huge load on the IDS. We included UNSW Argus/Bro feature set in our analysis for benchmarking purposes.

In the extraction step, we used the version provided by the authors and did not extract it manually. We took this decision mainly because of the complicated procedure needed to extract some of the features as explained earlier, in which, some parts are not publicly described. However, we supposed that the extraction costs are the highest among all sets. This is a realistic assumption due to the number of features needed.

Key			
srcIP	source IP address	Dpkts	Destination to source packet count
dstIP	destination IP address	swin	Source TCP window advertisement value
srcPort	source Port	dwin	Destination TCP window advertisement value
dstPort	destination Port	stcpb	Source TCP base sequence number
protocol	Protocol	dtcpb	Destination TCP base sequence number
Features			
srcIP	Source IP address	smeansz	Mean of the packet size transmitted by the src
sport	Source port number	dmeansz	Mean of the packet size transmitted by the dst
dstIP	Destination IP address	trans_depth	Represents the pipelined depth into the connection of http request/response transaction
dsport	Destination port number	res_bdy_len	Actual uncompressed content size of the data transferred from the server's http service.
proto	Transaction protocol	Sjit	Source jitter (mSec)
state	Indicates to the state and its dependent protocol, e.g. ACC, CLO, CON, ECO, ECR, FIN, INT, MAS, PAR, REQ, RST, TST, TXD, URH, URN, and (-) (if not used state)	Djit	Destination jitter (mSec)
dur	Record total duration	Stime	record start time
sbytes	Source to destination transaction bytes	Ltime	record last time
dbytes	Destination to source transaction bytes	Sintpkt	Source interpacket arrival time (mSec)
sttl	Source to destination time to live value	Dintpkt	Destination interpacket arrival time (mSec)
dttl	Destination to source time to live value	tcprrt	TCP connection setup round-trip time, the sum of 'synack' and 'ackdat'.
sloss	Source packets retransmitted or dropped	synack	TCP connection setup time, the time between the SYN and the SYN_ACK packets.
dloss	Destination packets retransmitted or dropped		
service	http, ftp, smtp, ssh, dns, ftp-data ,irc and (-) if not much used service		
Sload	Source bits per second		
Dload	Destination bits per second		
Spkts	Source to destination packet count		

Table 3.5: Features of the UNSW Bro/Argus set.

Features			
ackdat	TCP connection setup time, the time between the SYN_ACK and the ACK packets.	ct_dst_ltm	No. of connections of the same destination address (3) in 100 connections according to the last time (26).
is_sm_ips_ports	If source (1) and destination (3) IP addresses equal and port numbers (2)(4) equal then, this variable takes value 1 else 0	ct_src_ltm	No. of connections of the same source address (1) in 100 connections according to the last time (26).
ct_state_ttl	No. for each state (6) according to specific range of values for source/destination time to live (10) (11).	ct_src_dport_ltm	No of connections of the same source address (1) and the destination port (4) in 100 connections according to the last time (26).
ct_flw_http_mthd	No. of flows that has methods such as Get and Post in http service.	ct_dst_sport_ltm	No of connections of the same destination address (3) and the source port (2) in 100 connections according to the last time (26).
is_ftp_login	If the ftp session is accessed by user and password then 1 else 0.	ct_dst_src_ltm	No of connections of the same source (1) and the destination (3) address in 100 connections according to the last time (26).
ct_ftp_cmd	No of flows that has a command in ftp session.	attack_cat	The name of each attack category. In this data set, nine categories e.g. Fuzzers, Analysis, Backdoors, DoS Exploits, Generic, Reconnaissance, Shellcode and Worms.
ct_srv_src	No. of connections that contain the same service (14) and source address (1) in 100 connections according to the last time (26).		
ct_srv_dst	No. of connections that contain the same service (14) and destination address (3) in 100 connections according to the last time (26).		

Table 3.6: Features of the UNSW Bro/Argus set (continuous).

3.2 Extraction Ability for Encrypted Traffic

Inspite of the usage of an unencrypted dataset in this work, we would like to emphasize the ability of extraction of each feature set because, in the case of a real world IDS implementation, most of the traffic would be encrypted.

3.2.1 TLS

TLS (Transport Layer Security) is a cryptographic protocol that provides data integrity and privacy for a communication. Symmetric cryptography is used between two end points so that the communication is encrypted and both ends use secret keys generated from a shared key at the TLS handshake. The data transfer is reliable because of the message integrity check that is included with each transmission, which censures the manipulation of the data.

TLS runs on top of the transport layer in the OSI model. If a feature is extracted from a higher level than the transport layer, its extraction would not be feasible. In other words, the features derived from the IP header and the TCP/UDP/ICMP headers are reachable even with TLS but above this level, everything is encrypted.

3.2.2 IPsec

IPsec (Internet Protocol Security) is a cryptographic scheme that encrypts data sent over a network. IPsec is an end-to-end security scheme that negotiates encryption keys between end points also at the beginning of a session. The IPsec runs on top of the Internet layer and has many implementations. It can be used in transport mode (only the IP payload is encrypted) or in tunnel mode (the whole IP packet is encrypted and a new artificial header is appended at the beginning of the packet). The IPsec is then more severe and, hence, feature extraction will not be possible even for a lower layer protocols. For instance, TCP and UDP ports are not accessible with an IPsec implementation.

From the discussion above, conclusions are summarized in Table 3.7. In the table we can see the extraction ability of each feature set in both encryption cases.

Feature set	Flow identification	Flow extraction and classification
UNSW	TLS/–	–/–
CAIA	TLS/–	TLS/–
Consensus	TLS/–	TLS/IPsec
TA	TLS/–	TLS/IPsec
AGM	TLS/IPsec	TLS/–

Table 3.7: Traffic encryption and feature sets compatibility.

3.3 Preprocessing

Preprocessing was one of the major steps in this work. Starting by constructing the feature sets with their respective JavaScript Object Notation (JSON) files, uni-to-bidirectional flow mapping, labeling the data, nominal to numerical feature transformation, scaling the data and finally feature selection/reduction.

3.3.1 Feature Sets Construction

We constructed the four sets: TA, Consensus, CAIA and AGM using a tool developed by a member of our research group (see Appendix A). The tool takes *pcap* files, a couple of parameters and a JSON file that lists the different features and operations needed to represent a flow. For each set, we created a different JSON structure.

The output of the Flow Exporter was not complete in the sense that it did not represent the final version of each set. For instance, values in forward and backward directions were not grouped together for both Consensus and CAIA (unidirectional flows) and no aggregation existed for AGM vector. However, the TA vector was on its final version.

We created a script that executed the flow exporter command (see Appendix A) for each *pcap* and outputted a CSV file. The CSVs were merged later to form the final CSV of each set. Additionally, a timeout of 60 seconds was used for the extraction.

For a unique identification of flows that showed the same 5-tuple key, we additionally used the flow starting timestamp. This allowed us to differentiate flows having the same key but occurring in different times.

The AGM format needed packets aggregation; therefore, another script took care of configuring the Flow Exporter to output basic packet's header features (discussed earlier) from pcaps instead of flows. there was no proper definition of a timeout, but only an observation time window, which was chosen to be 900 seconds.

After having the outputs from the above steps (four CSV files), we proceeded to the next step.

Note that a python script was used to control the whole operation and keep track of the running time which was needed later for the evaluation.

3.3.2 Unidirectional to Bidirectional Flows & Aggregation

The resulting CSVs represented unidirectional flows (and packets for AGM). In order to convert unidirectional flows into bidirectional we had to find flows that belonged to the same connection and then merge the values by creating two sub-features. For instance, number of packets became number of packets in forward direction and number of packets in backward direction. The aggregation for the AGM was simple but consumed a considerable time.

We divide this section into two parts. First we talk about the uni-to-bidirectional transformation and, second, we discuss the aggregation step for the AGM set.

Unidirectional to Bidirectional mapping. The goal here was to convert the resulting (unidirectional) flows for both CAIA and Consensus into bidirectional traffic. We created a python script that loaded the whole data (for each set), took in consideration the flow key (5-tuple) and the flow starting timestamp. Then, the script merged the complete dataset with a copy of it with an inverted key i.e., if the original key was $\{srcIP_A, dstIP_B, srcPort_A, dstPort_B,$

```

{Flow_key = "srcIP_A, dstIP_B, srcPort_A, dstPort_B, Protocol", Flow_timestamp_start =
"00000000", Flow_time_stamp_end = "00000010", Flow_features = "featA1, featA2..."}

{Flow_key = "srcIP_B, dstIP_A, srcPort_B, dstPort_A, Protocol", Flow_timestamp_start =
"00000002", Flow_time_stamp_end = "000000xx", Flow_features = "featB1, featB2..."}

{Flow_key = "srcIP_A, dstIP_B, srcPort_A, dstPort_B, Protocol", Flow_timestamp_start =
min(A,B), Flow_time_stamp_end = max(A,B), Flow_features = "feat1_forward,
feat2_forward, feat1_backward, feat2_backward..."}

```




Figure 3.2: An example of mapping two unidirectional flows into one bidirectional flow.

Protocol} plus a starting time-stamp, the copy has the following key: $\{srcIP_B, dstIP_A, srcPort_B, dstPort_A, Protocol\}$, with the additional condition that the starting timestamp in the second dataset should be in the range of starting \rightarrow ending timestamp of the original dataset (the obvious reason was to keep only flows belonging to the same connection). Next, if a match was found, the script filled up a new table where it stored the key, the feature values from flow_A extra labeled as forward and features values from flow_B extra labeled as backward. If no match was found, it kept the original values and filled up the backward columns with 0. Figure 3.2 shows this operation.

Note that the script kept also track of running times for further evaluation.

Aggregation (AGM). The AGM needed a different step instead of uni-to-bidirectional mapping. The later keeps track of hosts activity so we had to aggregate flows or packets belonging to the same source IP addresses. The script used performed the following: first, scanned the whole dataset to find distinct source IP addresses, created a hashing table which stored these distinct values. Afterwards, it went line by line (for every source IP address) and filled up all possible values from the original AGM dataset. In other words, for each feature, it stored a list of all distinct observed values and applied the corresponding statistical transformation on that list as discussed in 3.1.3. The last step was repeated for all seven features. As regards to the last feature of the AGM (*pkts*), we simply kept track of the number of packets in the entire time window (belonging to the same source IP address).

From the discussion above, it is now obvious that the AGM vector required more memory and time.

Note that also here, the script kept track of the running time.

3.3.3 Labeling

Labeling the data carefully was really important since it has a direct impact on the classification results. We carried out the labeling for TA, CAIA and Consensus (AGM had a different approach) as follows: we first loaded the GT file provided with the dataset, then we matched flows from the GT and the datasets. The key used for the matching was the well known 5-tuple plus the starting timestamp of each flow. The timestamp is added here since the same key could refer to flows that occurred in different time periods. Additionally, the GT file did not have a timeout and, therefore, a single flow in the GT could correspond to multiple flows in the extracted data. We labeled all flows that had a starting timestamp in the range of starting \rightarrow ending timestamp of the GT file with the same label. Thus, even if the flow was divided into multiple flows because of the timeout, the new sub-flows had all the same label and no information lost happened.

Regarding labeling the AGM dataset, it is obvious that the labeling can not be carried out as previously described since an AGM entry (connected to one source IP address) might contain multiple flows that represent either benign or malicious behavior and thus, many labels. Our approach was to keep track of all flows belonging to a unique source IP address and decide the label according to the given labels from the GT file. For instance, let say that a host was identified in 20 flows, 15 of them were malicious and 5 were benign. One could say that this host tended to show malicious activity more than benign activity so it should be labeled as malicious (the host in this case could also be a victim). However, deciding whether a host is malicious or not is risky. We decided to label an entry as malicious if it contains a least one malicious flow. Note that this step did not need an extra time because the label depends only on the previous condition.

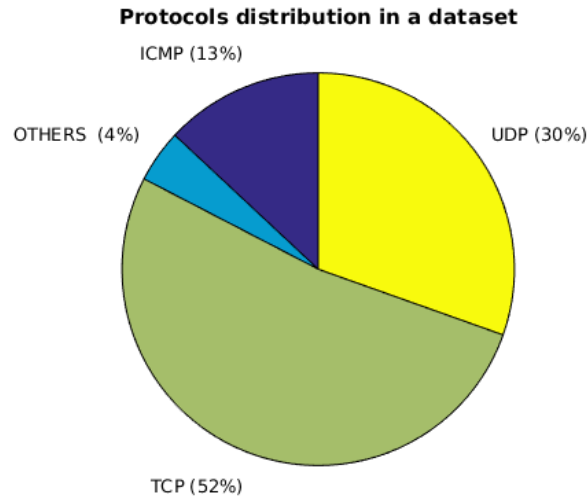
The labeling step created two extra columns in each CSV, one was of a binary type, i.e., attack or normal (1 or 0) and the other one was the attack type which represented the name of the attack in order to make deep analysis easier if needed (the attack category was also extracted from the GT file).

The UNSW Bro/Argus dataset was already labeled so this step was not necessary.

3.3.4 Nominal Features

Not all classification algorithms are capable to deal with nominal values. Most of the algorithms that are based on pure mathematical formulations need numerical data at the input and thus, by having categorical features, the classification algorithm will ignore them or outputs an error. However, these features might carry precious informations that might lead into better classification performance therefore discarding them should not be an option. Not only categorical features are not supported but also for instance: IP addresses and port numbers. Even though they are represented by numbers, performing mathematical transformations on them is illogical. A possible solution is using the so-called "dummy variables".

The first step of this approach is to create frequency tables of nominal features. Frequency tables show the number of times a precise and unique value occurs. A popular example is the computation of the frequency table for the protocol feature where most graphical representations would show three main peaks corresponding to TCP, UDP and ICMP respectively followed by other negligible peaks corresponding to all other protocols. The first three peaks can represent up to 90% of the traffic sometimes. We show next how the *Protocol* feature was replaced by three dummy features. Table 3.8 shows the new dummy features and Figure 3.3 shows the frequency plot of the Protocol feature. This approach for transforming traffic features is proposed in [7].



protocol	feat_1	feat_2	feat_3
TCP	0	0	1
UDP	0	1	0
ICMP	1	0	0
others	0	0	0

Figure 3.3 & Table 3.8: Nominal to numerical feature mapping example.

In what follows, we will extend this example to all our sets. Note however that in some cases the nominal features either have a very sparse distribution and thus, one can not convert them into hundreds of features or their relevance in the classification is negligible so they can discard. In this work, we proceed as follows (the same methodology was proposed in [7]): we created a frequency table for each nominal feature then, if the distribution was sparse, we ignored it otherwise we kept it for the analysis. Later, if the relevance of these dummy features was too low, we discarded them in the features selection step (Sections 3.3.6 and 3.3.7).

We discuss next details about the nominal features in each set.

TA and Consensus. In these two sets, the only nominal features were $\{srcPort, dstPort, Protocol\}$. We created frequency tables for each one of them. The *Protocol* feature got mapped into three dummy variables as expected. However, the *srcPort* and the *dstPort* showed a very sparse distribution. Actually, the first 5000 distinct *dstPort* represented only 72% of the total traffic and the

first 50000 distinct *srcPort* represented only 85% of the total traffic. It was then not feasible to create 5000 or 50000 new features only to keep the information carried by these two features (very sparse distribution will minimize the information gain so the feature will not carry any information for the classification procedure anyways and, also, the classification algorithms will build a very complex models to take all dummy variables in consideration).

CAIA. The CAIA set had only one nominal feature which was the *Protocol*. We replaced it with three dummy variables (as was shown in the example of Figure 3.3 and Table 3.8).

AGM. In the AGM, the mode feature gave always nominal values. We computed the frequency tables and we found: *mode_srcPort* the first 500 values represented 89% of the data thus, not feasible to map it. *Mode_dstPort* the first 500 values represented 75% of the data thus, also not feasible to map it. *Mode_protocol* had a condensed distribution so it was mapped into three dummy variables and finally *mode_TCPflag* also was mapped into three dummies. *mode_TTL* represented numerical values and was kept as it is.

Note that later it turned out that most of these dummy features had a negligible relevance for the classification and were hence discarded by the feature selection (the relevancy is discussed in Chapter 4).

UNSW Bro/Argus. The UNSW Bro/Argus set had also many nominal features. In addition to those which were present in TA and Consensus (same conclusion applied to them), this set had also *state* and *service* which needed to be converted into dummy variables. For the *state* feature, the two most frequent values represented more than 98% of the traffic (we talk here about FIN and CON states), thus, the *state* was replaced by two dummy variables. The second feature which was the *service* showed that the first seven values represented more than 99% of the traffic (nothing, dns, http, ftp-data, smtp, ssh and ftp-control) thus, we replaced it by seven dummy variables. The same conclusion as previously discussed applies here; the original features

were not relevant in the classification and thus the effort made here was not worth it.

One can notice that the source and destination IP addresses were neither used in the classification nor mapped into dummies, the simple reason behind was that the IP address did not contain any information and were only used to identify flows.

3.3.5 Scaling & Log-transformation

Scaling. An important step in the preprocessing was scaling the data and having it prepared for either a direct classification or further preprocessing techniques such as PCA. ML algorithms that use distance as basis for the classification are very sensitive to ranges and to the scale of the different dimensions (features). Without scaling, large features will always have a larger impact on the distance computed by the algorithm and smaller range features will not be relevant. It is then very important to bring down all ranges to the same order of magnitude. Geometrically speaking, unscaled features can be viewed as an n -dimensional space that has a non-uniform basis vectors, the space is stretched when features have large ranges and very tight when features have small ranges.

A few types of scaling are available, we cite:

1. Min-Max scaling: the data scaled this way will lay in the range $[0, 1]$.

The formula used is:

$$x_{scaled} = \frac{x_{original} - \min(x_{original})}{\max(x_{original}) - \min(x_{original})}$$

2. Mean normalization: this scaling will convert the data into a zero-mean data ($\mu = 0$). The corresponding formula is:

$$x_{scaled} = x_{original} - \mu$$

where μ is the mean of $x_{original}$.

3. Standardization: here, we remove the variance from the data and thus, it is a risky operation especially when dealing with Internet traffic (it was not used in this work). One should keep in mind that despite the benefits that a standardization would offer to some algorithms (SVM or Logistic Regression for instance), using it would cause a lost of information. However, we included it here for the sake of completeness. The formula is:

$$x_{scaled} = \frac{x_{original} - \mu}{\sigma}$$

where μ is the mean of $x_{original}$ and σ is the standard deviation.

In this work we combined 1 and 2. The first let the classification algorithms perform well without any bias and the second was necessary as a preparation of the data for the PCA. Many works in the past ignored these steps before performing PCA which would lead into a wrong transformation of data. This issue will be addressed in more details in Section 3.3.8.

Note that the dummy variables were not affected by these transformations for the simple reason that they are binary..

Log-transformation Basically, the log-transformation is used when a feature has a large range that is logarithmically distributed, this transformation will then fix this issue and makes the data again linearly distributed. However, the content and shape will both be changed since this transformation will cause a non-linear change of distances. In [4] the authors discussed the effects of this transformation deeply and concluded that it is better to avoid it.

The formula used for the log-transform is:

$$x_{transformed} = \log(1 + x_{original})$$

In this work, we tried the log-transformation in our experiments but it turned out that it had practically no effect. In addition and as said before, [4] discouraged to use it so it was not used.

3.3.6 Features Selection & Reduction

Some of the features are not relevant for the classification in the sense that they do not contribute to any decision, therefore, selecting which features to keep is really important. Feature selection might also be used to simplify the model that will be created by the ML algorithm, reduce training and classification times, avoid curse of dimensionality and solve the over-fitting problem. Many methods were proposed in the literature for an optimal selection. Each method uses different approaches, i.e., correlation, relevance, redundancy etc. In this work we opted for selecting features based on DT feature importance, which is explained in Section 3.3.7.

After finding the best parameters for a DT classifier (the parameters tuning is discussed in 3.4.2), the DT model computed feature importance. This was the basis for selecting features that had a non-zero importance. Results are discussed later in Chapter 4.

Another way of reducing dimension is features reduction. PCA can be used for this task, it would result in a set of a smaller dimension however, the new data will have some losses and the new features will have no practical or physical meaning, thus we used PCA only for decorrelating the data and not for features reduction. Note that PCA is based on variance maximization and is discussed in Section 3.3.8.

3.3.7 Features Importance via Decision Trees

When using DT or RF classifiers, an internal metric is calculated. It is called: *The Gini importance* and defined as follows:

$$I = G_{parent} - G_{split1} - G_{split2}$$

where G_i is the Gini index for a parent node or a children node that can be calculated as:

$$G = \sum_{i=1}^{n_c} p_i(1 - p_i)$$

here n_c represents the number of classes (binary in this work) and p_i is the ratio of classes in a tree branch. The sum of the *Gini importance* among all features results in a 1.

We will see later that DT and RF gave the best results in term of classification performance and, therefore, using them as a basis for the features selection is suitable.

3.3.8 Principal Component Analysis

One main drawback of having a large set of features is the so-called curse of dimensionality as discussed in [1] and [10]. We need to reduce the dimensionality as much as possible without any loss of information. Keeping a large number of features increases classification time cost and causes lower performances for some classifiers, especially if they include redundant and correlated features, i.e., Naive Bayesian, Neural Networks.

As discussed earlier, one of the most powerful methods used for features reduction is the well known PCA. It allows to construct a new representation of the data which maximizes the variance only in the first few new features without an acceptable degree of information loss.

After computing the covariance matrix of the data, PCA projects the original space into a rotated version that fits the data best with less dimensions. However, features in the new space have no meaning or physical interpretation, they are nothing but a linear combination of the original features. The eigen values of the covariance matrix tell us about the amount of information (or variance) contained in each new dimension.

In this work we used PCA to project the data into a newer space only to get rid of the correlation present on it which will help some classifiers perform better. Later, we again used DTs feature importance to remove features with zero relevancy. We could have also removed feature with a very low variance (corresponding to a very low eigenvalue) but we decided to keep 100 % of the data because feature selection already reduced the dimensionality and in any case, the dimensions in this work were not of a high order.

The key step before performing PCA was scaling the data and, most importantly, centering it (removing the mean). We already said that PCA is a rotation of all axis thus, working with a non-zero-mean data would cause a bias since the rotation center would not lay in the the origin (all-zeros coordinates) but at a point with different coordinates (the mean of each dimension). That is why the data need to be zero-mean before performing a PCA projection.

3.4 Supervised Analysis

In Chapter 2, we discussed the classification algorithms used in this work. These algorithms are used in a supervised manner. Supervised analysis are the type of analysis where one tries to predict a category or label under the assumption that one already know them. Obviously, since both original and predicted labels will be known, one can disclose how accurate and precise the resulting classification model is.

3.4.1 Training-Testing.

Our data was already labeled, so we proceeded as follows:

- First, we divided the entire dataset (for each feature set) into two parts, a training part and a testing part. The partition was not random but tried to keep the same ratio of normal-to-abnormal traffic distribution. The training set contained 80 % of the data where the testing set contained the other 20 %;
- once having the training and testing partitions, we train each algorithm first with the training data and the training labels. The algorithm would then build a model that fits the mapping: flow \Leftrightarrow label;
- once the model was built, we tested it with the testing data, which would then produce a prediction of the test labels;

- the predicted labels were compared with the original testing labels and afterwards metrics were computed (see Section 3.5).

Note that the script used to run all these operations was a bit more complex because it included a cross-validation step which was performed on the training part (discussed in Section 3.5.2).

3.4.2 Parameters Tuning

After deciding which ML classification algorithms to use in this work, we faced the problem of parameter tuning to get the best performance/overfitting trade-off. The methodology followed here was the so-called Grid Search which was utilized for each algorithm and each feature set. A grid search is a technique used to tune ML algorithms in order to maximize a certain metric. The main idea is as follows: assume a simple n -dimensional matrix where every dimension represents a parameter to tune. Next, consider a single dimension as a vector, so that the length and the values contained in this vector can be adjusted. The idea is to compute the metric that we want to maximize at each point and for all n -dimensions of the matrix. The output should be the tuple of parameters (coordinate in the n -dimensional space) that maximizes the metric. As an example: DT requires two parameters to be tuned : maximum depth and minimum number of leafs), therefore, a 2D matrix is created and later all 2D tuples contained are tested for the best output accuracy.

In this work, we tried to maximize the f1-score (it is discussed in Section 3.5.1). To make sure that the results were valid even for a different dataset, the maximization was carried out with an additional 5-fold cross-validation scheme.

A few things to note here: first, the Grid Search approach runs on pre-defined intervals. In this work, intervals were chosen first based on expert knowledge then enhanced depending on the results (scanning through a smaller range with smaller steps in each iteration). Second, some other

methodologies and algorithms are proposed in the literature for the parameters tuning. We cite the random search [3] and genetic algorithms [19].

We created a script that executes all the previous steps and outputs a list that contains the best parameters of all five algorithms for all feature sets. The tuning results are shown and discussed in Chapter 4.

3.5 Evaluation

The final step was the result comparison. For this purpose, we needed a couple of evaluation metrics, we will introduce the ones used further in the performance analysis and also comment on the over-fitting problem. We discuss how to solve it using the cross-validation.

3.5.1 Metrics

We needed to compare the predicted labels versus the original ones. Some metrics exist to achieve this goal but their meanings are different. We chose the five most popular metrics used for a classification evaluation; however, they are all derived from the *confusion matrix*, which is the matrix that summarizes classification results. The confusion matrix contains four fields for a binary classification: False Positives, False Negatives, True Positives and True Negatives. Figure 3.4 shows an example of a two-class confusion matrix.

		Original	
		0	1
Predicted	0	TN	FN
	1	FP	TP

Figure 3.4: The structure of a confusion matrix.

- True Positives (TP): when both labels –original and predicted by the classifier– are 1,

- True Negatives (TN): when both labels –original and predicted by the classifier– are 0,
- False Positives (FP): when the original label is 0 but the algorithm predicts a 1,
- False Negatives (FN): when the original label is 1 but the algorithm predicts a 0.

The five metrics derived from the confusion matrix and used in this work were:

- Accuracy : the accuracy gives a hint on the ratio of correctly classified data points. However, it is advised to avoid it when having unbalanced data. The accuracy is defined as :

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

- Precision : the precision gives a hint on the goodness of a classifier. In other words, it tells us to which degree we can trust the classifier's results. The precision is defined as following :

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

- Recall : the recall gives a hint on how good the classifier performs. In other words, how well it classifies data points. It is defined as :

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

- F1 score : the f1 score is not a metric by itself but a combination of Precision and Recall. It calculates the mean metric (harmonic mean) of both Precision and Recall and tends to be closer to the lowest values among the two. It can be defined as :

$$F1score = \frac{TP + TN}{TP + FP + FN + TN}$$

Or :

$$F1score = \frac{2 * Precision * Recall}{Precision + Recall}$$

Actually, this is why we tried to maximize the f1-score in the Grid Search method (to tune the parameters) because, by doing so, we were maximizing two metrics simultaneously.

- **Roc_Auc** : or, in other words, Area Under the Receiver Operation Characteristic curve. It represents the probability that the classifier will rank a randomly chosen positive sample higher than a randomly chosen negative sample.

3.5.2 Over-fitting Problem & 5-fold Cross-validation

Over-fitting. When training a model to fit some data, over-fitting occurs when the model fits the data so good that even the noise contained inside is taken into the decision rules. We can summarize what we just explained by saying: **a model that over-fits the data is not a robust model.**

Cross-validation. Cross-validation is a technique to enhance and validate the classification results. The idea is simple but computationally expensive. Let us assume that the dataset was already partitioned into testing and training sets. The next step would be to take the training part and divide it into "n" smaller sets, then, compute the previous metrics for each one of the "n" sets and average them. Finally, validate these results with the testing part (compare the values from both classifications –training/testing–). Figure 3.5 shows how this was done in the case of a 5-fold cross-validation where n = 5 in this case.

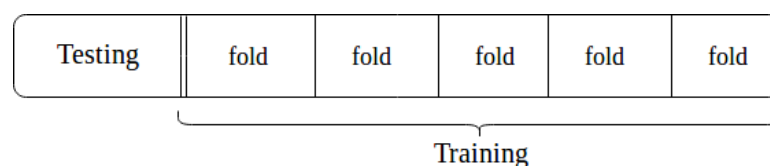


Figure 3.5: How a data set is divided in order to perform a 5-fold cross-validation.

A few things to note: first, dividing the training part into five folds needs to be done carefully such that the labels ratio should be kept the same among all folds (every part should have the same number of normal/abnormal labels). Second, averaging the scores of all folds is not a complete representation of the results, that is why we have to add the standard deviation. Third, in our case, the experiments were repeated also for the training part, this was mainly done to ensure that the model had the same performances for both training as well as testing parts which was another confirmation that no over-fitting occurred. The second and third points here are discussed further in depth in [Chapter 4](#).

Chapter 4

Results & Discussion

This chapter summarizes the evaluation of results. Experiments discussed in Chapter 3 were conducted in a machine with the following characteristics :

- *4x Intel(R) Core(TM) i5-4300 CPU @ 1.90GHz*
- *Memory: 4GB*
- *OS: Ubuntu 16.04 LTS*
- *Kernel: Ubuntu 4.13.0-37 generic*

The evaluation was divided into two parts: extraction times performance and classification performance. Note that a 5-fold cross-validation was used during the classification in order to validate our results. From all these results, we conducted a global comparison and a specific individual analysis.

4.1 Extraction Costs Performance

The extraction performance of each feature set can be evaluated using different metrics. In this work, we chose to evaluate it using the time needed to extract every set (time cost).

4.1.1 Time costs

The time cost was recorded simultaneously when the scripts were running. We tried our best to extract all feature sets by keeping the same extraction

methodology; however, some feature sets needed extra steps. These steps are summarized as follows:

- **Extraction:** this operation is directly influenced by the Flow Extractor. We expected however that the time performance would depend strongly on the number of features to extract. This step was performed for TA, Consensus, AGM and CAIA.
- **Uni-to-Bidirectional mapping:** the CAIA and Consensus sets needed this extra step because of their bidirectional nature.
- **Aggregation:** the AGM set needed this step because it aggregates flows in order to represent a single host behavior rather than presenting 5-tuple flows.
- **Labeling:** we labeled TA, Consensus, AGM and CAIA using the GT file as explained in Chapter 3.

The UNSW Bro/Argus feature set was not evaluated in terms of time costs. When inspecting its features and the extraction process, we assumed that this set was more costly than the other sets. Beyond that, authors do not provide a complete description of the extraction procedure in the corresponding papers [14] and [13].

Figure 4.1 shows a graphical representation of the time costs of all five sets (the UNSW Bro/Argus has an estimated curve). The exact values are shown in Table 4.1. These results were actually expected. We summarized them as follows :

Feature set	Extraction	Uni to Bidirectional	Aggregation	Labeling
UNSW	> 1h	<i>n/a</i>	–	<i>n/a</i>
CAIA	46m 34s	04m 21s	–	02m 21s
Consensus	41m 49s	03m 26s	–	02m 07s
TA	41m 32s	–	–	02m 01s
AGM	58m 37s	–	57m 16s	29m 17s

Table 4.1: Time cost in numbers.

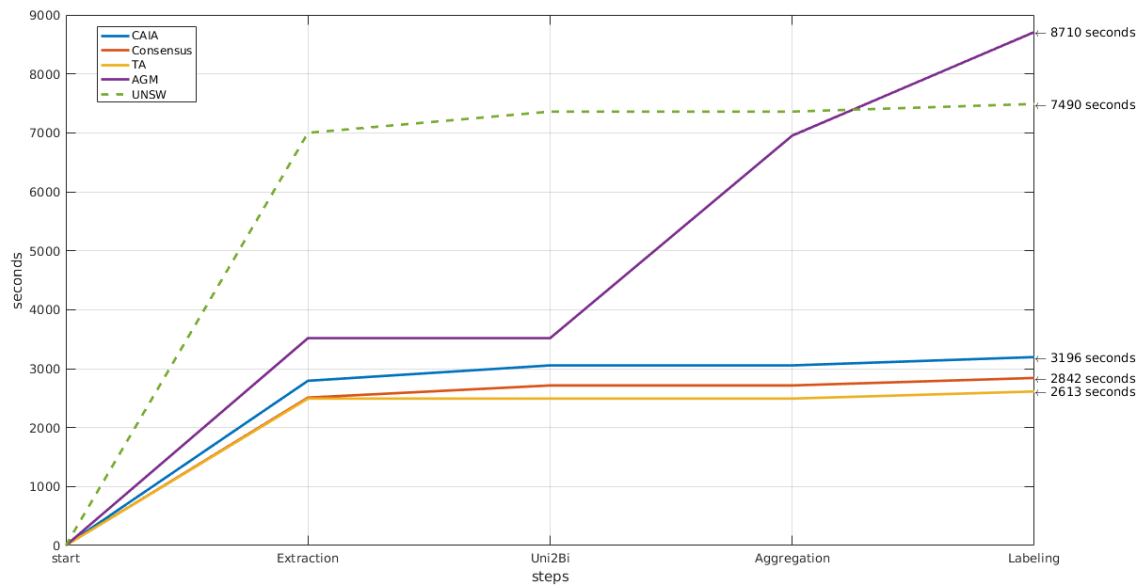


Figure 4.1: Time cost of each set. The UNSW set performance is only an estimation and it can differ.

Extraction

- TA and Consensus showed approximately the same extraction time. This is due to the same number of features that they contained and the similarities between them.
- CAIA had a longer extraction time because it involved more features than TA and Consensus.
- AGM had even a longer extraction time because hash structures based on only IP source address as key are much more complex than when using the 5-tuple as key. Even though the number of features is small, the aggregated packets per flow is are much higher.
- UNSW Bro/Argus was assumed to have the largest extraction time. This assumption was based on the nature of its features. UNSW Bro/Argus requires deeper inspection of traffic than the other options and uses 12 extraction methods that are not publicly available.

Uni-to-Bidirectional mapping

- There was a slight difference between CAIA and Consensus due only to the number of features needed to be converted into bidirectional mode;
- UNSW Bro/Argus also required this step. However, by manually inspecting the features and considering the overlapping with the CAIA vector, the time needed to perform the conversion would definitely be larger, therefore, it was set to the highest value again.

Aggregation

- This step was necessary only in the case of the AGM vector, which consumed notably more time.

Labeling

- CAIA, Consensus, TA and UNSW required the same time for labeling. The labeling strategy explained in Section 3.3.3 was the same for all of them;
- AGM needed a longer time due to its different labeling approach and the complex algorithmic structure. (details are explained in Section 3.3.3).

To summarize, in terms of extraction time costs, the TA vector was the fastest –it has less features and less steps were needed to extract it– the AGM was the slowest –the aggregation and labeling consumed more time and also memory– and the UNSW Bro/Argus set required a longer extraction time but then required a similar time cost as CAIA, TA and Consensus in other steps; so intuitively, it was safe to assume that it needs longer extraction time.

Regarding memory consumption, all sets had practically the same memory consumption. The later was proportional to the number of features used except AGM which needed more memory to store the hashing tables during the aggregation step.

4.2 Training-Testing Classification Performance

The next step is the evaluation of the classification performance. As described in Chapter 3, we used the algorithms with the five sets and performed a 5-fold cross-validation. Later, we computed a couple of metrics to compare classification results. Both training and test subsets were evaluated to ensure the nonexistence of over-fitting.

4.2.1 Time Activity vector

We started the comparison with the TA vector. A model was built using the best parameters extracted from the grid search and then used to predict the class of flows in the test set. The classification results are depicted in Table 4.2.

algorithm	train/test	accuracy	precision	recall	f1score	roc_auc
DT	training	0.988 (+/- 0.009)	0.798 (+/- 0.228)	0.835 (+/- 0.031)	0.811 (+/- 0.111)	0.983 (+/- 0.004)
DT	test	0.986 (+/- 0.009)	0.781 (+/- 0.221)	0.798 (+/- 0.030)	0.785 (+/- 0.116)	0.967 (+/- 0.008)
NB	training	0.968 (+/- 0.008)	0.634 (+/- 0.465)	0.613 (+/- 0.065)	0.356 (+/- 0.176)	0.831 (+/- 0.005)
NB	test	0.970 (+/- 0.007)	0.633 (+/- 0.476)	0.523 (+/- 0.061)	0.397 (+/- 0.109)	0.827 (+/- 0.010)
L2	training	0.973 (+/- 0.012)	0.762 (+/- 0.653)	0.422 (+/- 0.070)	0.301 (+/- 0.293)	0.840 (+/- 0.009)
L2	test	0.971 (+/- 0.013)	0.867 (+/- 0.680)	0.412 (+/- 0.069)	0.298 (+/- 0.296)	0.842 (+/- 0.010)
RF	training	0.988 (+/- 0.010)	0.786 (+/- 0.235)	0.851 (+/- 0.049)	0.811 (+/- 0.118)	0.994 (+/- 0.004)
RF	test	0.987 (+/- 0.009)	0.778 (+/- 0.224)	0.825 (+/- 0.034)	0.796 (+/- 0.116)	0.990 (+/- 0.005)
SVM	training	0.805 (+/- 0.182)	0.123 (+/- 0.457)	0.133 (+/- 0.378)	0.040 (+/- 0.058)	0.446 (+/- 0.199)
SVM	test	0.634 (+/- 0.411)	0.083 (+/- 0.143)	0.709 (+/- 0.543)	0.112 (+/- 0.097)	0.727 (+/- 0.293)
kNN	training	0.987 (+/- 0.009)	0.788 (+/- 0.228)	0.805 (+/- 0.101)	0.790 (+/- 0.111)	0.980 (+/- 0.007)
kNN	test	0.985 (+/- 0.009)	0.772 (+/- 0.210)	0.767 (+/- 0.088)	0.765 (+/- 0.112)	0.974 (+/- 0.023)

Table 4.2: TA classification results.

From these results, we can emphasize:

- Training and test results of each algorithm were almost the same, this confirmed that the prediction model did not over-fit the training set;
- the standard deviation which represents the cross-validation or the stability results showed good values for DT, RF and kNN. SVN, NB and LR2 had lower performance, these are deterministic problems that are

related to the dataset used which leads into high sensitivity when using ML algorithms;

- the three algorithms that performed the best overall are DT, RF and kNN;
- SVM and LR2 show the worst performances.

The parameters used for each of the classifiers are shown in Table 4.3. These parameters were obtained using the parameters tuning strategy discussed in Section 3.4.2.

DT	NB	L2	RF	SVM	kNN
min_samp_leaf = 2	alpha = 0.1	C = 11.421	min_samp_leaf = 2	C = 2000	k = 5
max_depth = 23		Tol = 0.003	max_depth = 23	gamma = 1000	
criterion = "gini"			criterion = "gini"	max_iter = 400	
			number_of_trees = 7		

Table 4.3: The optimal parameters for the TA classifiers.

4.2.2 Consensus vector

The consensus set classification results are shown in Table 4.4. The results indicated the following :

- Training and test results of each algorithm were almost the same, this confirmed that the prediction model did not over-fit the training set;
- the standard deviation in the case of SVM showed lower performance, this means that SVM has deterministic problems and high sensitivity; however, this problem was already addressed in the TA results. Other algorithms had a lower standard deviation which corresponded to a stable model;
- the three algorithms that performed the best are DT, RF and kNN;
- SVM, NB and LR2 show the worst performances, especially SVM.

The parameters used for each classifier are shown in Table 4.5. These parameters were obtained using the parameters tuning strategy discussed in Section 3.4.2.

algorithm	train/test	accuracy	precision	recall	f1score	roc_auc
DT	training	0.989 (+/- 0.011)	0.820 (+/- 0.227)	0.882 (+/- 0.052)	0.847 (+/- 0.143)	0.995 (+/- 0.004)
DT	test	0.989 (+/- 0.008)	0.849 (+/- 0.216)	0.826 (+/- 0.058)	0.832 (+/- 0.088)	0.994 (+/- 0.005)
NB	training	0.972 (+/- 0.023)	0.631 (+/- 0.378)	0.650 (+/- 0.025)	0.625 (+/- 0.198)	0.964 (+/- 0.016)
NB	test	0.972 (+/- 0.024)	0.625 (+/- 0.384)	0.646 (+/- 0.031)	0.621 (+/- 0.211)	0.962 (+/- 0.015)
L2	training	0.969 (+/- 0.008)	0.621 (+/- 0.316)	0.266 (+/- 0.019)	0.366 (+/- 0.050)	0.982 (+/- 0.018)
L2	test	0.969 (+/- 0.007)	0.615 (+/- 0.313)	0.248 (+/- 0.034)	0.346 (+/- 0.043)	0.980 (+/- 0.018)
RF	training	0.991 (+/- 0.006)	0.887 (+/- 0.177)	0.844 (+/- 0.030)	0.862 (+/- 0.076)	0.998 (+/- 0.003)
RF	test	0.990 (+/- 0.008)	0.864 (+/- 0.217)	0.851 (+/- 0.046)	0.853 (+/- 0.091)	0.998 (+/- 0.003)
SVM	training	0.657 (+/- 0.725)	0.196 (+/- 0.516)	0.452 (+/- 0.874)	0.099 (+/- 0.058)	0.686 (+/- 0.092)
SVM	test	0.567 (+/- 0.864)	0.422 (+/- 0.738)	0.476 (+/- 0.833)	0.112 (+/- 0.118)	0.622 (+/- 0.072)
kNN	training	0.990 (+/- 0.007)	0.861 (+/- 0.196)	0.842 (+/- 0.035)	0.848 (+/- 0.086)	0.986 (+/- 0.004)
kNN	test	0.989 (+/- 0.007)	0.849 (+/- 0.194)	0.828 (+/- 0.033)	0.835 (+/- 0.091)	0.984 (+/- 0.008)

Table 4.4: Consensus classification results.

DT	NB	L2	RF	SVM	kNN
min_samp_leaf = 1	alpha = 47.076	C = 94.789	min_samp_leaf = 1	C = 100	k = 5
max_depth = 9		Tol = 0.0009	max_depth = 9	gamma = 1000	
criterion = "gini"			criterion = "gini"	max_iter = 400	
			number_of_trees = 10		

Table 4.5: The optimal parameters for the Consensus classifiers.

4.2.3 AGM vector

The AGM set classification's results are shown in Table 4.6. From these results, a few things to note :

- The training and test results sometimes significantly differ, this is due to the number of simples used in this set (as we already discussed, AGM aggregates flows thus, its size was much smaller than other sets);
- the standard deviation which represents only the cross-validation or the stability results showed good values for RF, LR2 and NB. kNN, SVN

and DT had poor results. This time, it was DT who has deterministic problems;

- the two algorithms that showed the best performance are RL2 and RF;
- generally speaking, all algorithms excluding NB showed good performance; however, sometimes the standard deviation was too large which may lead into a non-consistent results.

The parameters used for each classifier (after tuning) are shown in Table 4.7.

algorithm	train/test	accuracy	precision	recall	f1score	roc_auc
DT	training	0.991 (+/- 0.014)	0.985 (+/- 0.080)	0.899 (+/- 0.194)	0.936 (+/- 0.110)	0.988 (+/- 0.038)
DT	test	0.964 (+/- 0.068)	0.772 (+/- 0.535)	0.778 (+/- 0.617)	0.753 (+/- 0.532)	0.993 (+/- 0.027)
NB	training	0.982 (+/- 0.033)	0.858 (+/- 0.296)	0.944 (+/- 0.099)	0.892 (+/- 0.179)	0.990 (+/- 0.026)
NB	test	0.985 (+/- 0.036)	0.867 (+/- 0.303)	1.000 (+/- 0.000)	0.921 (+/- 0.183)	0.993 (+/- 0.020)
L2	training	0.995 (+/- 0.009)	0.979 (+/- 0.086)	0.955 (+/- 0.084)	0.966 (+/- 0.062)	0.996 (+/- 0.016)
L2	test	0.993 (+/- 0.022)	0.983 (+/- 0.125)	0.939 (+/- 0.247)	0.954 (+/- 0.154)	0.999 (+/- 0.009)
RF	training	0.995 (+/- 0.011)	0.989 (+/- 0.057)	0.949 (+/- 0.147)	0.967 (+/- 0.086)	0.997 (+/- 0.022)
RF	test	0.988 (+/- 0.036)	0.973 (+/- 0.200)	0.889 (+/- 0.398)	0.910 (+/- 0.288)	0.995 (+/- 0.022)
SVM	training	0.997 (+/- 0.006)	0.995 (+/- 0.038)	0.961 (+/- 0.084)	0.977 (+/- 0.044)	0.994 (+/- 0.037)
SVM	test	0.997 (+/- 0.017)	1.000 (+/- 0.000)	0.961 (+/- 0.201)	0.977 (+/- 0.118)	0.997 (+/- 0.022)
kNN	training	0.998 (+/- 0.006)	1.000 (+/- 0.000)	0.966 (+/- 0.083)	0.982 (+/- 0.043)	0.994 (+/- 0.030)
kNN	test	0.998 (+/- 0.012)	1.000 (+/- 0.000)	0.978 (+/- 0.166)	0.987 (+/- 0.100)	0.989 (+/- 0.083)

Table 4.6: AGM results.

DT	NB	L2	RF	SVM	kNN
min_samp_leaf = 6	alpha = 0.1	C = 11.421	min_samp_leaf = 6	C = 0.1	k = 5
max_depth = 7		Tol = 0.003	max_depth = 7	gamma = 10	
criterion = "gini"			criterion = "gini"	max_iter = 400	
			number_of_trees = 12		

Table 4.7: AGM classifiers optimal parameters.

4.2.4 CAIA vector

The CAIA set classification results are shown in Table 4.8. These results are practically similar to those of Consensus, discussed earlier in Section 4.2.2.

However, CAIA showed slightly better performances, which is intuitive due to the overlapping between the sets on one hand, and more features in CAIA on the other hand. Note that the same conclusions in Consensus regarding classification performance apply to CAIA.

The parameters used for each classifier are shown in Table 4.9. These parameters were obtained using the parameters tuning strategy discussed in Section 3.4.2.

algorithm	train/test	accuracy	precision	recall	f1score	roc_auc
DT	training	0.990 (+/- 0.006)	0.882 (+/- 0.176)	0.827 (+/- 0.028)	0.851 (+/- 0.076)	0.976 (+/- 0.008)
DT	test	0.989 (+/- 0.008)	0.843 (+/- 0.229)	0.834 (+/- 0.054)	0.833 (+/- 0.098)	0.975 (+/- 0.011)
NB	training	0.967 (+/- 0.021)	0.526 (+/- 0.227)	0.731 (+/- 0.015)	0.605 (+/- 0.154)	0.957 (+/- 0.014)
NB	test	0.967 (+/- 0.022)	0.526 (+/- 0.228)	0.730 (+/- 0.031)	0.604 (+/- 0.161)	0.957 (+/- 0.017)
L2	training	0.979 (+/- 0.020)	0.695 (+/- 0.343)	0.817 (+/- 0.092)	0.735 (+/- 0.178)	0.983 (+/- 0.013)
L2	test	0.975 (+/- 0.019)	0.676 (+/- 0.361)	0.682 (+/- 0.089)	0.661 (+/- 0.159)	0.981 (+/- 0.015)
RF	training	0.991 (+/- 0.006)	0.888 (+/- 0.177)	0.848 (+/- 0.031)	0.864 (+/- 0.075)	0.998 (+/- 0.003)
RF	test	0.990 (+/- 0.008)	0.860 (+/- 0.224)	0.855 (+/- 0.054)	0.852 (+/- 0.095)	0.998 (+/- 0.004)
SVM	training	0.554 (+/- 0.696)	0.031 (+/- 0.041)	0.400 (+/- 0.813)	0.047 (+/- 0.028)	0.465 (+/- 0.063)
SVM	test	0.881 (+/- 0.327)	0.309 (+/- 0.544)	0.296 (+/- 0.383)	0.211 (+/- 0.185)	0.689 (+/- 0.124)
kNN	training	0.990 (+/- 0.007)	0.866 (+/- 0.194)	0.844 (+/- 0.035)	0.851 (+/- 0.086)	0.985 (+/- 0.006)
kNN	test	0.989 (+/- 0.007)	0.853 (+/- 0.200)	0.832 (+/- 0.049)	0.838 (+/- 0.092)	0.984 (+/- 0.008)

Table 4.8: CAIA classification results.

DT	NB	L2	RF	SVM	kNN
min_samp_leaf = 5	alpha = 59.167	C = 100	min_samp_leaf = 5	C = 100	k = 5
max_depth = 21		Tol = 0.0009	max_depth = 21	gamma = 100	
criterion = "gini"			criterion = "gini"	max_iter = 400	
			number_of_trees = 16		

Table 4.9: The optimal parameters for the CAIA classifiers.

4.2.5 UNSW Argus/Bro vector

Finally, we discuss the UNSW set classification results. They are in Table 4.10. These results showed in general a really good performance similar to that of CAIA and Consensus. In the case of the SVM classifier, there was even an improvement thus, one can conclude that large dimensions help SVM performs

better, the reason behind is the better representability of the data when using more dimensions which leads into a reduction of the inconsistency present at the edge of the decision regions.

The parameters used for each classifier are shown in Table 4.11. These parameters were obtained using the parameters tuning strategy discussed in Section 3.4.2.

algorithm	train/test	accuracy	precision	recall	f1score	roc_auc
DT	training	0.989 (+/- 0.007)	0.864 (+/- 0.194)	0.831 (+/- 0.023)	0.844 (+/- 0.086)	0.971 (+/- 0.007)
DT	test	0.988 (+/- 0.008)	0.850 (+/- 0.209)	0.823 (+/- 0.044)	0.832 (+/- 0.096)	0.960 (+/- 0.013)
NB	training	0.975 (+/- 0.006)	0.775 (+/- 0.195)	0.376 (+/- 0.079)	0.505 (+/- 0.104)	0.985 (+/- 0.007)
NB	test	0.975 (+/- 0.006)	0.782 (+/- 0.192)	0.371 (+/- 0.084)	0.502 (+/- 0.106)	0.985 (+/- 0.007)
L2	training	0.986 (+/- 0.014)	0.790 (+/- 0.311)	0.869 (+/- 0.081)	0.816 (+/- 0.155)	0.997 (+/- 0.005)
L2	test	0.986 (+/- 0.014)	0.791 (+/- 0.309)	0.861 (+/- 0.085)	0.813 (+/- 0.150)	0.996 (+/- 0.006)
RF	training	0.991 (+/- 0.006)	0.874 (+/- 0.168)	0.871 (+/- 0.018)	0.870 (+/- 0.081)	0.998 (+/- 0.002)
RF	test	0.990 (+/- 0.007)	0.858 (+/- 0.186)	0.873 (+/- 0.030)	0.862 (+/- 0.088)	0.998 (+/- 0.003)
SVM	training	0.967 (+/- 0.003)	0.768 (+/- 0.443)	0.046 (+/- 0.116)	0.078 (+/- 0.154)	0.775 (+/- 0.214)
SVM	test	0.968 (+/- 0.010)	0.635 (+/- 0.414)	0.251 (+/- 0.066)	0.353 (+/- 0.113)	0.795 (+/- 0.108)
kNN	training	0.988 (+/- 0.008)	0.846 (+/- 0.222)	0.818 (+/- 0.037)	0.827 (+/- 0.095)	0.988 (+/- 0.003)
kNN	test	0.987 (+/- 0.009)	0.829 (+/- 0.238)	0.807 (+/- 0.057)	0.812 (+/- 0.101)	0.988 (+/- 0.005)

Table 4.10: UNSW Bro/Argus classification results.

DT	NB	L2	RF	SVM	kNN
min_samp_leaf = 6	alpha = 0	C = 100	min_samp_leaf = 6	C = 100	k = 5
max_depth = 22		Tol = 0.0009	max_depth = 22	gamma = 1	
criterion = "gini"			criterion = "gini"	max_iter = 400	
			number_of_trees = 23		

Table 4.11: The optimal parameters for the UNSW Bro/Argus classifiers.

4.3 Global comparison

All the previous results can be gathered together, we discuss them if a few points that are presented in the following.

- The Consensus set showed the best extraction-classification trade-off. For a light IDS it might be the best choice;
- if preprocessing speed is one of the priorities besides the classification performances, the TA set is the best choice in this case;
- if we do not care about the speed but we want a really accurate classification, AGM or UNSW Argus/Bro are the ones to go for. AGM is preferable with encrypted traffic but it does not provide much details about the attack (because of the aggregation). UNSW Bro/Argus provides detailed information about the attack but can not be extracted from encrypted traffic, so another trade-off raises here;
- CAIA is an enhanced version of the Consensus. If we want a slightly better classification results on the price of a small extra time, CAIA is the one to choose.

4.4 Features importance

In this section, we analyzed each feature set separately. We try to determine how every feature impacts the decision of each classifier. Note, however, that feature importance is not valid or has no meaning when features are isolated (analysis only with a single feature). In our case, feature importance is mainly valid in the context of the whole set.

We classified the features along the following scale : highly relevant features, relevant, low relevance and negligible, Figure 4.2 depicts this scale.

4.4.1 Time Activity vector

- **Highly relevant** : bytes (13.9), pkts (17.1), bytes_per_seconds-active (50.4), interval (10.1)
- **Relevant** : pkts_per_secondsactive (5.6), maxtoff (1.1), protocol:TCP (1.1)

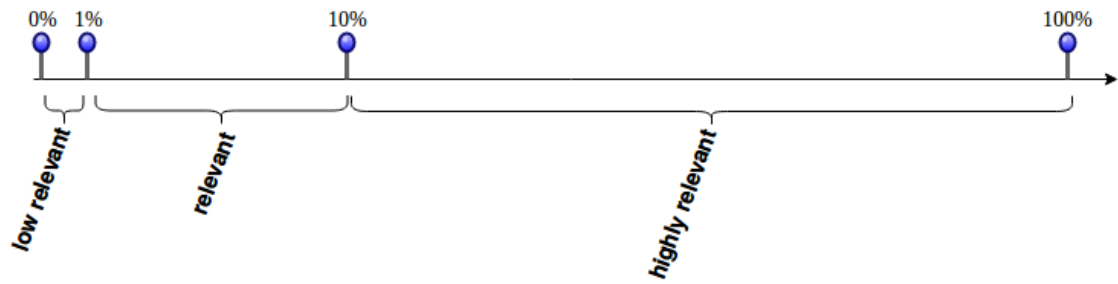


Figure 4.2: The importance scale used to classify the features. 0 importance is classified as negligible.

- **Low relevance** : maxton (0.4), minton (0.1), protocol:UDP (0.1)
- **Negligible** : seconds-active (0.0), mintoff (0)

4.4.2 Consensus vector

- **Highly relevant** : min_dstPktLength (75.5)
- **Relevant** : dstBytes (6.9), max_srcPktLength (6.2), median_srcPktLength (5.4), max_dstPktLength (2.2)
- **Low relevance** : srcBytes (0.9), duration (0.9), min_srcPktLength (0.9), protocol:UDP (0.2), median_srcPktIAT (0.2), mode_srcPktLength (0.1), variance_srcPktIAT (0.1), variance_dstPktIAT (0.1)
- **Negligible** : srcPkts (0.1), dstPkts (0.0), median_dstPktIAT (0.0), mode_dstPktLength (0), protocol:TCP (0)

4.4.3 AGM vector

- **Highly relevant** : #pktLength (85.2)
- **Relevant** : pkts_mode_srcPort (2.1), pkts_mode_dstPort (6.1), #protocol (2.7), pkts_mode_pktLength (3.7)
- **Low relevance** : mode_pktLength (0.1)
- **Negligible** : #dstIP (0), mode_dstIP (0), pkts_mode_dstIP (0), #srcPort (0), mode_srcPort (0), #dstPort (0), mode_dstPort (0), mode_protocol

(0), pkts_mode_protocol (0), #TTL (0), mode_TTL (0), pkts_mode_TTL (0), #TCPflag (0), mode_TCPflag (0), pkts_mode_TCPflag (0), pkts (0)

4.4.4 CAIA vector

- **Highly relevant** : min_dstPktLength (74.8)
- **Relevant** : max_srcPktLength (7.1), dstBytes (6.1), mean_srcPktLength (2.3), max_dstPktLength (2.0), protocol:tcp (1.7), stdev_srcPktLength (1.5), mean_dstPktLength (1.1), duration (1.1)
- **Low relevance** : srcBytes (0.5), min_srcPktLength (0.5), mean_dstPktIAT (0.3), mean_srcPktIAT (0.2), stdev_dstPktLength (0.2), stdev_dstPktIAT (0.2), #srcTCPflag:syn (0.2), stdev_srcPktIAT (0.1)
- **Negligible** : srcPkts (0.0), dstPkts (0.0), max_srcPktIAT (0.0), protocol:UDP (0.0), #srcTCPflag:fin (0.0), #srcTCPflag:ack (0.0), #dstTCPflag:ack (0.0), #dstTCPflag:fin (0.0), max_dstPktIAT (0.0), min_srcPktIAT (0), min_dstPktIAT (0), #srcTCPflag:cwr (0), #dstTCPflag:syn (0), #dstTCPflag:cwr (0)

4.4.5 UNSW Argus/Bro vector

- **Highly relevant** : ct_state_TTL (80.0), ct_srv_dst (11.1)
- **Relevant** : dstMeansz (1.7), dstBytes (1.4), srcBytes (1.2)
- **Low relevance** : srcWin (0.7), srcMeansz (0.5), service:http (0.4), dstPkts (0.3), ct_dst_src_ltm (0.3), dstIntpkt (0.3), synack (0.3), service:other (0.2), duration (0.2), srcTTL (0.1)
- **Negligible** : service:ftp (0.0), dstLoss (0.0), srcLoad (0.0), dstLoad (0.0), srcPkts (0.0), srcTcpcb (0.0), dstTcpcb (0.0), res_bdy_len (0.0), srcJit (0.0), dstJit (0.0), srcIntpkt (0.0), tcprtt (0.2), ackdat (0.0), ct_flw_http_mthd (0.0), ct_srv_src (0.8), ct_src_ltm (0.0), ct_src_dport_ltm (0), dstTTL (0), srcLoss (0), dstWin (0), trans_depth (0), is_sm_ips_ports (0), is_ftp_login

(0), ct_ftp_cmd (0), ct_dst_ltm (0), ct_dst_sport_ltm (0), state:CON (0), state:FIN (0), state:INT (0), service:dns (0), service:ftp-data (0), service:smtp(0), service:ssh (0)

We can see some remarkable details that are present in all sets. For instance, features related to packet's length show a high relevance in all sets. This was somehow self explanatory because most of the time, the packet's length of a malicious packet is static and thus represents a footprint of an attack (research already disclosed this phenomena before so these results were expected). Another thing that we noticed is that different statistical transformations of the "number of packets" feature creates sub-features with a modest relevance.

Constructing a new set based on best features from each set does not necessarily mean the construction of a perfect set because the relevancy scale was relative to the set itself. Highly relevant features in a set can be negligible in another. This is due to many reasons; the most intuitive one is that if two features are highly correlated and grouped together in the same set, one of them will for sure govern the relevance scale and the other will be negligible.

Chapter 5

Conclusions

5.1 Conclusions

In this work we compared five feature vectors recently proposed and used for traffic classification and anomaly detection in communication networks. We conducted experiments to study the performance of each feature set in terms of preprocessing costs and classification accuracy. In order to accomplish this task, a labeled dataset containing 100GB of modern Internet traffic was used (Appendix B). The used dataset is the UNSW-NB15, which mixes legitimate and illegitimate traffic which correspond to 9 different families of attacks. Six well known ML algorithms were used for the classification tasks : Naive Bayes classifiers, Decision Trees, Random Forests, Support Vector Machines, l2-Logistic Regression and k-Nearest Neighbors.

The comparison was mainly intended to obtain insights about different ways of representing network traffic. For instance, the TA feature set represents the unidirectional footprint of Internet traffic activities, CAIA groups popular statistical features in a bidirectional manner, Consensus is similar to CAIA but constructed based on a meta-analysis of the most used features in the recent literature, UNSW Bro/Argus is based of Bro and Argus tools, which carry out a complicated and a deep packets inspection methodology, and finally the AGM, which focuses on analyzing and the aggregation of sources. By doing so, we compared actually five different "types" of sets.

The conducted experiments lead to three main conclusions:

1. The AGM feature set showed the best classification results, therefore,

profiling hosts instead of connections (flows) may reveal some attacks categories better. However, due to its extraction complexity, its implementation is therefor challenging.

2. The feature set constructed by performing meta-analysis on past research (Consensus) showed the best classification/extraction performance trade-off. This revealed the following related facts: when using a flow features based set, first, not all statistical features are relevant (the case of CAIA feature set). Second, not all possible network traffic features are needed (the case of UNSW Bro/Argus feature set) and finally monitoring flows in a bidirectional manner will improve the performance (the TA feature set was unidirectional) since it offers more information about the connection.
3. The most relevant feature in approximately all sets was the *packet's length* and its different statistical transformations. This implies that, when constructing a new set, it might be beneficial to use more features based on *packet's length*. This however does not mean that the *packet's length* is capable of classifying traffic by itself, other features are still needed.

5.2 Submitted Publication

A paper entitled “**Analysis of Lightweight Feature Vectors for Attack Detection in Encrypted Network Traffic**” was derived from this work. In this paper, we compared the five feature sets by evaluating the time costs and the classification performance only using the RF classifier since it showed the best results. We also covered traffic encryption and tried to study the extraction capability of each set before and after the features selection step.

The paper has been submitted as to the IEEE International Conference on Data Mining series (ICDM 2018) and is currently under review.

5.3 Further Work

Based on this work, many questions arise and need to be answered. Examples of further work are:

- Further research can study and compare the usage of these feature sets in the case of unsupervised classification.
- New combinations of statistical transformations can likewise be applied to the basic features and their relevancy should be studied. This can really affect classification results since a good transformation might reveal hidden information carried by a raw feature.
- Enhanced feature sets can be created based on the calculated features relevancy. Then, tested first with the same dataset, for which good results are expected. And later tested with a different new dataset in order to have a general and final validation.
- With deep analysis, the effect of the ML learning algorithms on the classification can also be studied in depth. This mean: Does a certain algorithm always misclassify a certain attack or a kind of traffic and why?

Appendix A

Extraction Tool : Flow Extractor

The Flow Exporter is a tool developed in Go language by one of the **Communication Networks Group**'s members, at the **Technische Universität Wien**. Its main goal is to extract information from traffic captures (pcaps for instance) with the possibility of providing different outputs i.e.,: flows, packets etc.

By specifying which features to extract (defined by IANA [9]), the Flow Exporter takes in our case a pcap file and outputs a CSV file. The selection of the set of features needed is done using a JSON file. Few parameters also need to be set for a correct operation, for example : timeout, maximum active time, type (flows or packets) etc. The usage syntax is depicted in the following listing :

Usage:

```
go-flows [flags] callgraph features [featureargs] spec.json
        [features ...] [export type [exportargs]] [export ...]
        [...]
```

Usage:

```
go-flows [flags] callgraph [args] -spec commands.json
```

Writes the resulting callgraph in dot representation to stdout.

Featuresets (features), outputs (export), and, optionally, sources need to

be provided. It is possible to specify multiple feature statements and multiple export statements. All the specified exporters always export the features specified by the preceding feature group.

At least one feature specification and one exporter is needed

Identical exporters can be specified multiple times. Beware, that those will share a common exporter instance, resulting in a field set specification per specified featureset, and mixed field sets (depending on the feature specification).

Instead of providing the commands on the command line, it is also possible to use a json file.

A list of supported exporters and features can be seen with the list command. See also `go-flows callgraph features -h`.

Examples:

```
Export the feature set specified in example.json to example.csv
go-flows callgraph features example.json export csv
example.csv input [input ...]
```

```
Export the feature sets a.json and b.json to a.csv and b.csv
```

```
go-flows callgraph features a.json export csv a.csv
features b.json export b.csv input [input ...]
```

Export the feature sets a.json and b.json to a single common.csv (this results in a csv with features from a in the odd lines , and features from b in the even lines)

```
go-flows callgraph features a.json features b.json export
common.csv input [input ...]
```

Execute the commands provided in commands.json

```
go-flows callgraph -spec commands.json [...]
```

Flags:

```
-blockprofile string
    Write goroutine blocking profile
-cpuprofile string
    Write cpu profile
-heapprofile string
    Write heap profile
-memprofile string
    Write memory profile
-memprofilerate int
    Set MemProfileRate
-mutexprofile string
    Write mutex blocking profile
-trace string
    Turn on tracing
```

Args:

```
-active uint
    Active timeout in seconds (default 1800)
```

```

-expire uint
    Check for expired timers with this period in seconds.
    (default 100)
-filter string
    Process only packets matching specified bpf filter
-idle uint
    Idle timeout in seconds (default 300)
-n uint
    Number of parallel processing tables (default 4)
-perpacket
    Export one flow per Packet
-size uint
    Maximum packet size read from source. 0 = automatic (
    default 9000)
-spec string
    Load exporters and features from specified json file
-stats
    Output statistics

```

The complete syntax used for TA, Consensus and CAIA is :

```

go-flows offline -idle 60 -active 60 features input.json export csv
output.csv input input.pcap

```

and for the AGM :

```

go-flows offline -perpacket features AGM.json export csv output.csv
input.pcap

```

A.1 Structure

The structure of the flow extractor is depicted in Figure [A.1](#)

The state of the art of the Flow Extractor is described as follows, The tool start by creating hashing tables that contains all features provided via the JSON file. Next, by exploring the pcaps, for each packet, it checks the flow

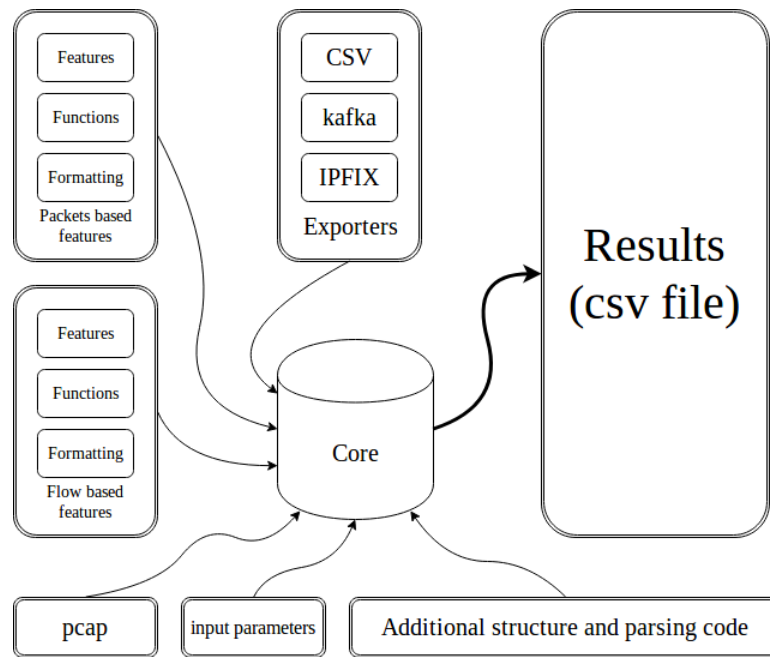


Figure A.1: A simplified structure of the Flow Exporter.

key and execute one of two functions, if the flow is new, it execute the *Start* function or if the flow exist already, it executes the *Event* function. these two functions are executed for each feature provided. At the end (timeout), it executes the *End* function. Now let us explain what these three functions represent:

- **Start function** : This function is called each time a new flow is detected. It consists basically of some initializations of internal variables. An example of this function is the following :

```
func (f *count) Start(context EventContext) {
f.count = 0
}
```

The exact syntax is not important for us but we can see an initiation of the variable *f.count* (OOP is used here).

- **Event function** : This function is called if a flow is already created and a new packet with the same flow key is identified. Basically this function tells the tool what to do with the specific feature value in the identified flow. An example of it is the following :

```
func (f *count) Event(new interface{}, context EventContext, src
interface{}) {
    f.count++
}
```

Here we can see that it is a simple incrementation of the variable *f.count*. Note that the syntax can be more complicated for instance if we use a value from the packet itself (for instance packet length or record time) and perform operations on it.

- **End function** : The end function is called when the flow ends (by a timeout for instance) and exports the final results performed on the internal variables. An example is :

```
func (f *count) Stop(reason FlowEndReason, context EventContext)
{
    f.SetValue(f.count, context, f)
```

Here we are outputting the final value of the counter that we already incremented on each packet arrival. More complex operations can be done (for instance calculate a mean or a statistical property of a feature).

By grouping the three functions together, the count feature function is constructed, which simply calculates the number of packets in a flow.

The complete syntax is more complicated and is not explained in this work.

A.2 Performance

We chose the Flow Exporter in this work for three reasons :

- First, because it is written in Golang where the execution speed is much larger than most of the tools available in public;
- second, because it is easy to use and features can be constructed based on needs;

- third, because it is developed by one of our team members and thus, the need to promote it :-)

When comparing the Flow Extractor's performance to other tools developed in other languages (Java, python...) we found out that the Flow Exporter in Go language outperforms them by far.

Note : the Flow Exporter is not on its full version and thus, not referenceable. It is available for private usage only and will be available to public later.

Appendix B

The NUSW-NB15 Dataset.

The raw network packets (Pcap files) of the UNSW-NB 15 data set is 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 activities. The UNSW-NB15 source files are provided in different formats, Pcap files, BRO files, Argus Files and CSV files. The source files of the data set were divided based in the date of the simulation 22-1-2015 and 17-2-2015, respectively.

The creation procedure and more statistics can be found in [13] and [14].

B.1 Statistics

Traffic type	Count	Attacks	Count
0 - legitimate	2218755	fuzzers	24246
1 - attacks	321283	reconnaissance	13987
total	2540038	shellcode	1511
		analysis	2677
		ackdoor	2329
		DoS	16353
		exploits	44525
		generic	215481
		worms	321283

Table B.1: Normal to Abnormal distribution of flows.

Table B.2: Attacks distribution.

Bibliography

- [1] Charu C Aggarwal, Alexander Hinneburg, and Daniel A Keim. “On the surprising behavior of distance metrics in high dimensional space”. In: *International conference on database theory*. Springer. 2001, pp. 420–434.
- [2] James P Anderson. “Computer security threat monitoring and surveillance”. In: *Technical Report, James P. Anderson Company* (1980).
- [3] James Bergstra and Yoshua Bengio. “Random search for hyper-parameter optimization”. In: *Journal of Machine Learning Research* 13.Feb (2012), pp. 281–305.
- [4] FENG Changyong et al. “Log-transformation and its implications for data analysis”. In: *Shanghai archives of psychiatry* 26.2 (2014), p. 105.
- [5] Daniel C Ferreira et al. “A meta-analysis approach for feature selection in network traffic research”. In: *Proceedings of the Reproducibility Workshop*. ACM. 2017, pp. 17–20.
- [6] Félix Iglesias and Tanja Zseby. “Analysis of network traffic features for anomaly detection”. In: *Machine Learning* 101.1-3 (2015), pp. 59–84.
- [7] Félix Iglesias and Tanja Zseby. “Pattern Discovery in Internet Background Radiation”. In: *IEEE Transactions on Big Data* (2017).
- [8] Félix Iglesias and Tanja Zseby. “Time-activity footprints in IP traffic”. In: *Computer Networks* 107 (2016), pp. 64–75.
- [9] *Internet Assigned Numbers Authority*. URL: <https://www.iana.org/>.
- [10] Eamonn Keogh and Abdullah Mueen. “Curse of dimensionality”. In: *Encyclopedia of Machine Learning and Data Mining*. Springer, 2017, pp. 314–315.

-
- [11] Yeon-sup Lim et al. "Internet traffic classification demystified: on the sources of the discriminative power". In: *Proceedings of the 6th International Conference*. ACM. 2010, p. 9.
- [12] Fares Meghdouri, Félix Iglesias, and Tanja Zseby. "Analysis of Lightweight Feature Vectors for Attack Detection in Encrypted Network Traffic". In: (2018). Pending.
- [13] Nour Moustafa and Jill Slay. "The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set". In: *Information Security Journal: A Global Perspective* 25.1-3 (2016), pp. 18–31.
- [14] Nour Moustafa and Jill Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)". In: *Military Communications and Information Systems Conference (MilCIS), 2015*. IEEE. 2015, pp. 1–6.
- [15] *scikit-learn, Machine Learning in Python*. URL: <http://scikit-learn.org/stable/>.
- [16] Charles J Stone. "Classification and regression trees". In: *Wadsworth International Group* 8 (1984), pp. 452–456.
- [17] Alina Vlăduțu, Dragoș Comănesci, and Ciprian Dobre. "Internet traffic classification based on flows' statistical properties with machine learning". In: *International Journal of Network Management* 27.3 (2017).
- [18] Nigel Williams, Sebastian Zander, and Grenville Armitage. "A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification". In: *ACM SIGCOMM Computer Communication Review* 36.5 (2006), pp. 5–16.
- [19] Alden H Wright. "Genetic algorithms for real parameter optimization". In: *Foundations of genetic algorithms*. Vol. 1. Elsevier, 1991, pp. 205–218.
- [20] Jun Zhang et al. "Robust network traffic classification". In: *IEEE/ACM Transactions on Networking (TON)* 23.4 (2015), pp. 1257–1270.