USING ENSEMBLE LEARNING FOR THE NETWORK INTRUSION DETECTION PROBLEM

Roland Mpoyi Kalonji (1755336)
Supervisor: Dr Jacoba Bührmann

School of Mechanical, Industrial and Aeronautical Engineering
University of the Witwatersrand
Johannesburg, South Africa

A dissertation submitted to the Faculty of Engineering and the Built Environment, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Master of Science in Engineering.

UNIVERSITY OF THE
WITWATERSRAND,
JOHANNESBURG

August 1, 2019

CANDIDATE'S DECLARATION

I declare that this dissertation is my own unaided work. It is being submitted for the Degree of Master of Science to the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination to any other university.

Signature:  Date: 01/08/2019

ABSTRACT

Nowadays, most organizations and platforms employ an intrusion detection system (IDS) to enhance their network security and protocol systems. The IDS has therefore become an essential component of any network system; it is a tool with several applications that can be tuned to specific content in a network by identifying various accesses (normal or attack). However, network intrusion detection system (NIDS) that focuses on revealing suspicious activities, is not effective in solving various problems such as identifying false IP packets and encrypted traffic. Hence, this work investigates the use of ensemble learning to solve these types of network intrusion detection problems (NIDPs). Random forest (RF), Decision Tree (DT) and Support Vector Machine (SVM) are introduced as classifiers based on Boruta and Principal Component Analysis (PCA) algorithms. In general, the main difficulties in using ensemble for the intrusion problem are to minimize false alarms and to maximize detection accuracy (Anuar et al., 2008). Additionally, the NIDP is divided into five categories, namely the detection of probe attacks, denial of service, remote to local, user to root and normal instances. Each problem is examined by one of the three aforementioned classifiers. In tackling these problems, the three classifiers achieved competitive results comparing to the works conducted by Balon-Perin (2012), Zainal et al. (2009) and Kevric et al. (2017). The results revealed that ensemble learning achieved more than 99% accuracy in demarcating attacks from normal connections. Particularly, RF, DT and SVM allowed to safeguard the NIDS from known and unknown attacks by developing reliable techniques. The KDD99 and NSL KDD datasets have been used to implement and measure the system performance (Fan et al., 2000; Dhanabal and Shantharajah, 2015).

Keywords: ensemble learning, network intrusion detection problem, random forest, decision tree, support vector machine.

ABRÉGÉ

De nos jours, la plupart des organisations et plates-formes utilisent le système de détection d'intrusion (IDS) pour améliorer la sécurité de leur réseau et leurs systèmes de protocole. L'IDS est donc devenu un composant essentiel de tout système de réseau. C'est un outil avec plusieurs applications qui peuvent être ajustés à un contenu spécifique d'un réseau en identifiant divers accès (normal ou attaque). Cependant, le système de détection d'intrusion sur le réseau (NIDS), qui, lui se focalise sur la révélation d'activités suspectes, n'est pas efficace pour résoudre divers problèmes, tels que l'identification de faux paquets IP et le trafic chiffré. Par conséquent, ce travail étudie l'utilisation de l'apprentissage d'ensemble pour résoudre ces types de problèmes de détection d'intrusion réseau (NIDP). Les forêts aléatoires (RF), les arbres de décision (DT) et les vecteurs de support (SVM) sont introduits en tant que classificateurs basés sur les algorithmes de Boruta et de l'analyse en composantes principales (PCA). En général, les principales difficultés liées à l'utilisation de l'ensemble pour le problème d'intrusion sont de minimiser les fausses alarmes et d'optimiser la précision de la detection (Anuar et al., 2008).. En outre, le NIDP est divisé en cinq catégories, à savoir la détection des attaques par sonde, le déni de service, les instances distantes par rapport aux utilisateurs locaux, les utilisateurs par défaut et les utilisateurs normaux. Chaque problème est examiné par l'un des trois classificateurs susmentionnés. En s'attaquant à ces problèmes, les trois classificateurs ont obtenu des résultats compétitifs par rapport aux travaux de Balon-Perin (2012), Zainal et al. (2009) ainsi que Kevric et al. (2017). Les résultats ont révélé que l'apprentissage d'ensemble permettait d'obtenir une précision de plus de 99% dans la démarcation des attaques des connections normales. En particulier, RF, DT et SVM ont permis de protéger le NIDS contre des attaques connues et inconnues en développant des techniques fiables. Les jeux de données KDD99 et NSL KDD ont été utilisés pour mettre en œuvre et mesurer les performances du système (Fan et al., 2000; Dhanabal and Shantharajah, 2015).

Mots-clés: apprentissage d'ensemble, problème de détection d'intrusion réseau, forêt aléatoire, arbres de décision, machine à vecteurs de support.

DEDICATION

Dedicated to my parents Jean Kalonji and Rosy Kalonji for their financial support, constant encouragement and prayer.

ACKNOWLEDGEMENTS

I am deeply indebted to many people for the roles they played in producing this work. Special thanks goes to:

1. Dr J.Bührmann, my supervisor, for her help throughout the compilation of this dissertation. She made valuable suggestions for the use of ensemble learning and for the implementation of the system developed in RStudio.
2. Mr Brett Freeman (Associate Lecturer, School of Mechanical, Industrial and Aeronautical Engineering, University of the Witwatersrand) for his contributions to this work.
3. Jean Kalonji, my father, for all of his help on machine learning classifiers which allowed me to reconsider areas I had overlooked.
4. Vitale, Deborah, Abbondah, Mulgra and Trinel Kalonji.
5. Mr Mulalo (4th year student, Wits-Mechanical Engineering) for proofreading this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

Acronym	Meaning
ACCS	Australian Centre of Cyber Security
ADFA-LD	Australian Defence Force Academy-Linux Dataset
ADTree	Alternating Decision Tree
AI	Artificial Intelligence
AIDS	Application-based intrusion detection system
ANN	Artificial Neural Network
ARP	Address Resolution Protocol
AttAcc	Attack Accuracy
AUC	Area Under the Curve
AvgAcc	Average Accuracy
BPNN	Back Propagation Neural Network
BPSO	Binary Particle Swarm Optimization algorithms
Caret	Classification and regression training
DARPA	Defence Advanced Research Projects Agency
DDoS	Detection of distributed Denial of Service
DNS	Domain Name System
DoS	Denial of Service
DPU	Data Pre-processing Unit
DR	Domain Rating
Dst port	Destination Port
DT	Decision Tree
ELM	Extreme Learning Machine
FAR	False Acceptance Rate
FN	False Negative
FP	False Positive
FTP	File Transfer Protocol
GA	Genetic Algorithm
HADS	Host-based Anomaly Detection System
HIDS	Host-based Intrusion Detection System

IMAP	Internet Message Access Protocol
IP address	Internet Protocol address
IQR	Interquartile range
KDD	Knowledge discovery and Data mining Dataset
K-NN	K-Nearest Neighbors algorithm
MAC	Media Access Control
MFM	Mean F-Measure
ML	Machine Learning
mRMR	minimum Redundancy Maximum Relevance
MZSA	Maximum Z score among shadow attributes
NBTree	Naive Bayes Tree
NIDS	Network Intrusion Detection System
NIDP	Network Intrusion Detection Problem
NIPS	Network-based Intrusion Prevention System
Nmap	Network Mapper
Ntree	Number of trees
OOB	Out-Of-Bag
OS	Operating System
PCA	Principal Component Analysis
R2L	Remote to Local
RBF	Radial Basis Function
RF	Random Forest
ROC curves	Receiver Operating Characteristic curves
RST	Rough Set Technique
SA	Simulated Annealing neural network
SMTP	Simple Mail Transfer Protocol
SOM	Self-Organizing Maps
SPARCstations	Scalable Processor Architecture stations
Src port	Source Port
Srv	Service record
SVM	Support Vector Machine
TCP/IP	Transmission Control Protocol/Internet Protocol
TN	True Negative

TP	True Positive
TPR	True Positive Rate
Tf × idf	Term Frequency × Inverse Document Frequency
U2R	User to Root
UNM	University of New Mexico
WD	Windows Dataset
Z-score	Standard score

EPIGRAPH

I have learned that success is to be measured not so much by the position that one has reached in life as by the obstacles which he has had to overcome while trying to succeed (…)

-Booker T. Washington-

Chapter 1-Introduction & Background

To raise new questions, new possibilities, to regard old problems from a new angle, requires creative imagination and marks real advance in science…

-Albert Einstein-

In recent years, network systems have captured the attention of many researchers. The number of webpages and growing amount of data available on the Internet have increased and expose the analysts to various challenges; for instance, the IPv4 (Internet Protocol version 4) address exhaustion and scarcity (Levin and Schmidt, 2014). Particularly, the Internet has been used as an important component for business models (Shon and Moon, 2007). For business activities, businesses and customers use the Internet applications such as websites and e-mails for their daily activities. Therefore, the security of information when using the Internet must be carefully considered. Intrusion detection is a major research topic when looking at personal and professional networks.

Initially, an intrusion can be thought of as any set of actions that attempts to compromise the integrity, confidentiality, or availability of a resource (Debar et al., 2000). Intrusion Detection Systems (IDSs) are software that detect, identify and respond to unauthorized or abnormal activities on a target system (Chen et al., 2005). NIDS involves looking for unusual, abnormal, unexpected and interesting data patterns (Shoemaker and Hall, 2011). NIDSs can be divided into two categories: anomaly and misuse (signature) detection based on different approaches. On one hand, anomaly detection attempts to determine whether a connection from the established normal usage patterns can be reported as an intrusion. On the other hand, misuse detection uses known attack patterns or system weaknesses to identify intrusions (Anderson, 1995). The terms IDS and NIDS are used interchangeably in the literature. In this work, it is referred to as NIDS for consistency.

The first paper about NIDSs is dated on 26th February 1980 and was written by James P. Anderson (1980). The topic was "Computer Security Threat: Monitoring and Surveillance". James Anderson (1980) divided attacks into external and internal ones based on whether the user had access permission to the computer or not. Then the main targets of the security audit trail were the following (Anderson, 1980):

- detecting internal attacks and unusual behaviour of users of resources;
- identifying the attacker strategy; enough data should be obtained in order for the security administrators to identify the problem.

Moreover, when an intruder attempts to break into an information system, tries to access confidential information or performs an action not legally allowed within a network, this is called the Network Intrusion Detection Problem (NIDP) (Graham, 2000). There are various systems designed to identify Internet-based intrusions. Particularly, Network Intrusion Detection Systems (NIDSs) allow to resist external attacks. However, NIDS that focuses on revealing suspicious activities, is not effective in solving various problems such as identifying false IP packets, encrypted traffic and unknown attacks. It is then beneficial to integrate Machine Learning (ML) based-ensemble classifiers into the network security by improving the NIDS performance. Hansen and Salamon (1990) proved that the combination of various classifiers could significantly enhance the overall accuracy of the predictions (Hansen and Salamon, 1990). Hence, this work describes the use of ensemble classifiers which provides the analysts with a set of rules and strategies when building the NIDS.

1.1 Problem statement

The NIDP is a complex and very broad problem due to the following aspects:

- Categories of network attack, computer security and systems.
- Varieties in OSs (Operating systems) and applications software.
- Variations into information sources, detection mechanisms and deployment approaches, etc.

The main problem NIDS battle with is the inability to identify new attacks and encrypted malicious traffic. This study focuses on anomaly-based intrusion detection systems and signature-based intrusion detection systems (Anderson, 1995 ; Shoemaker and Hall, 2011). Based on these two approaches, this study aims to detect new attacks, minor variations of known instances and to reduce false alarms. Thus, the features that distinguish normal data from abnormal data and false positive (FP) rate must be identified and significantly reduced; ML techniques that facilitate the process of demarcating normal data from abnormal data must be investigated. Most ML based-ensemble use tools and well known techniques like combining class labels, continuous output, etc. (Polikar, 2012). However, these techniques are not very effective for classifying data as normal or abnormal with great precision. It is therefore necessary to customize strong classifiers based on various aspects and requirements for the NIDS; the quality of detection is therefore of paramount importance. This implies that ensemble classifiers must be accurate and very efficient in encountering malicious traffic with a high detection rate.

1.2 Objectives and Research question

1.2.1 Objectives

The successful implementation of NIDS based-ensemble classifiers requires several factors to be considered. The main objective of this study is to propose a framework, test and analyse the NIDS. The sub-objectives are as follows:

1. Implement efficient algorithms to address the problem of intrusion detection.
2. Implement a real-time detection analysis system.
3. Improve the NIDS's accuracy and compare results to the work conducted by Balon-Perin (2012) and Zainal et al. (2009).

1.2.2 Research question

Central research question:

Is ensemble learning a viable technique to address the NIDP and to improve the accuracy?

1.3 Motivation and Importance of the research

1.3.1 Motivation

In recent years, ensemble classifiers have shown greater flexibility than single technique and are thus gaining popularity. It is even more likely to build a NIDS based-ensemble learning. Moreover, NIDS based-ensemble is attractive and has been motivated because it:

- allows to analyze normal and abnormal activities;
- is efficient in detecting network traffic as normal or attack;
- performs well when the data is very huge;
- classifies unknown instances;
- combines multiple classifiers; and
- enhances the overall accuracy and attack detection of the NIDS.

Although ML has been widely applied in various fields, there is not much research on NIDS based-ensemble classifiers (Chih-Fong et al., 2009).

Figure 1 illustrates yearwise distribution of ML studies in terms of classifiers design into NIDSs; the number of papers using single, hybrid and ensemble method between 2000 and 2007 (Chih-Fong et al., 2009). The number of papers on using ensemble classifier is very limited.

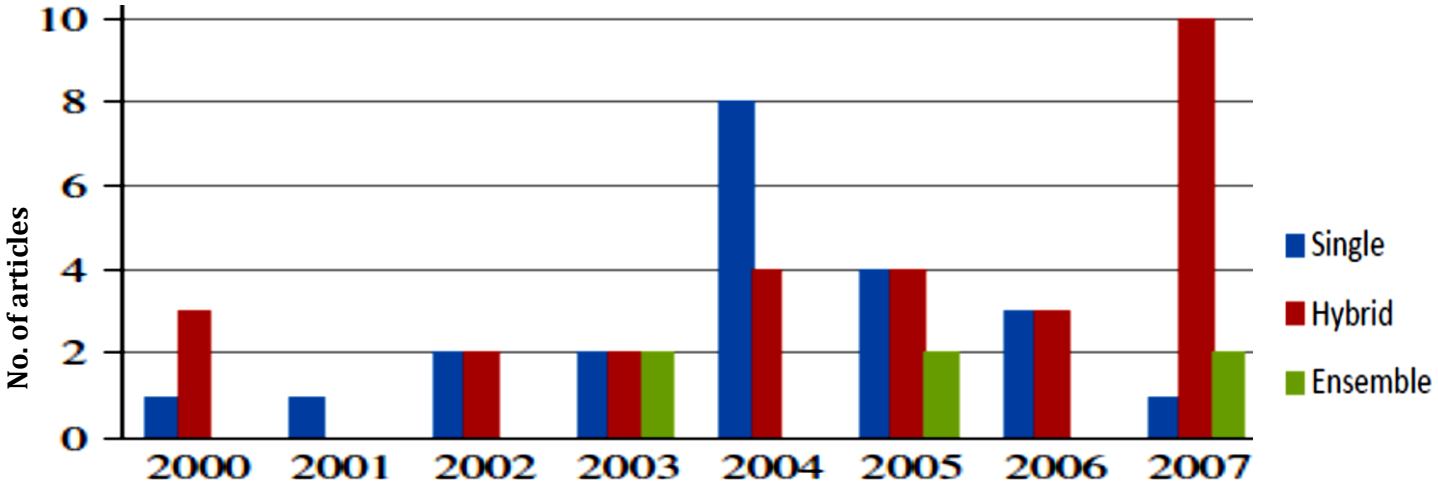


Figure 1 Yearwise distribution of articles for the types of classifier design (Chih-Fong et al., 2009).

Figure 2 presents the distribution of papers by year of publication based on different classifiers designs (between 2009 and 2014):

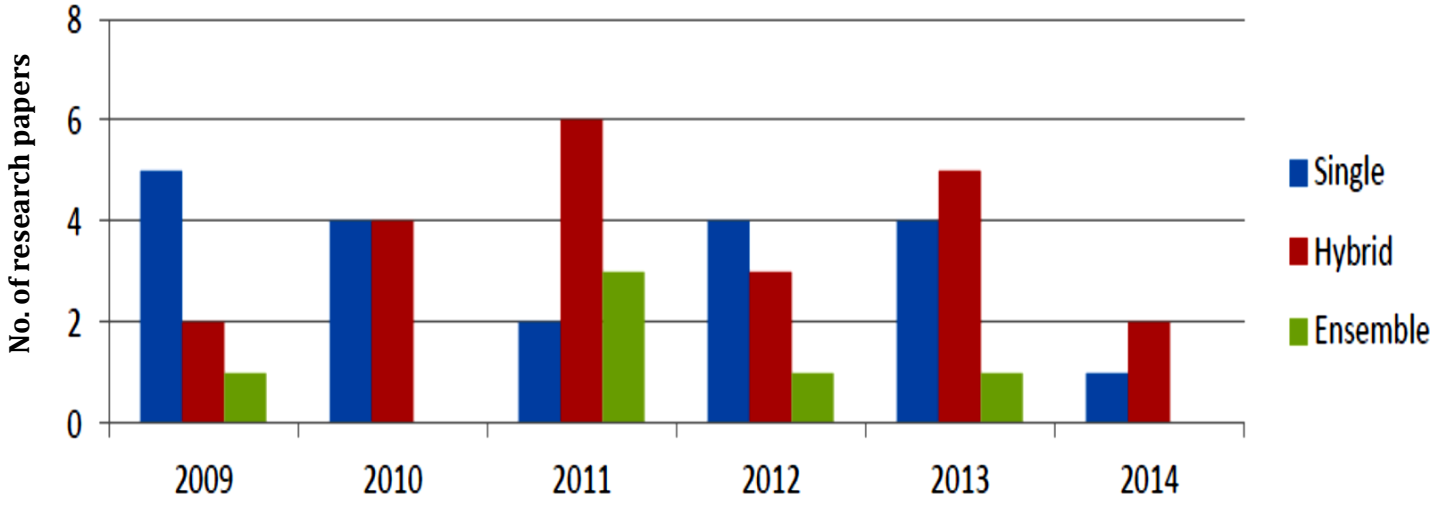


Figure 2 Yearwise distribution of papers (Haq et al., 2015).

Figure 1 and 2 show that ensemble technique has not been fully documented. Hence this research will use three classifier, ensemble based-methods and feature selection algorithms in order to construct a number of decision function highly accurate with low classification error.

1.3.2 Importance of the research

This section covers the importance of this dissertation in general. Initially, the objectives stated in section 1.2.1 allow to understand the advantages and limits of the NIDS based-ensemble techniques. The main challenges faced by ensemble techniques have been presented by Sommer and Paxson (2010). Compared to other areas in which ML has been applied, NIDS appears to be challenging; it requires a high level of accuracy. Additionally, the NIDS based-ensemble would have difficulty in identifying attacks where instances are different from previously observed data. Although ML is an appropriate technique to identify known attack variants, day-to-day vulnerability detection may be out of reach for a single "ML" technique. The lack of labelled data sets is also a huge problem for the NIDS based-ensemble (Sommer and Paxson, 2010).

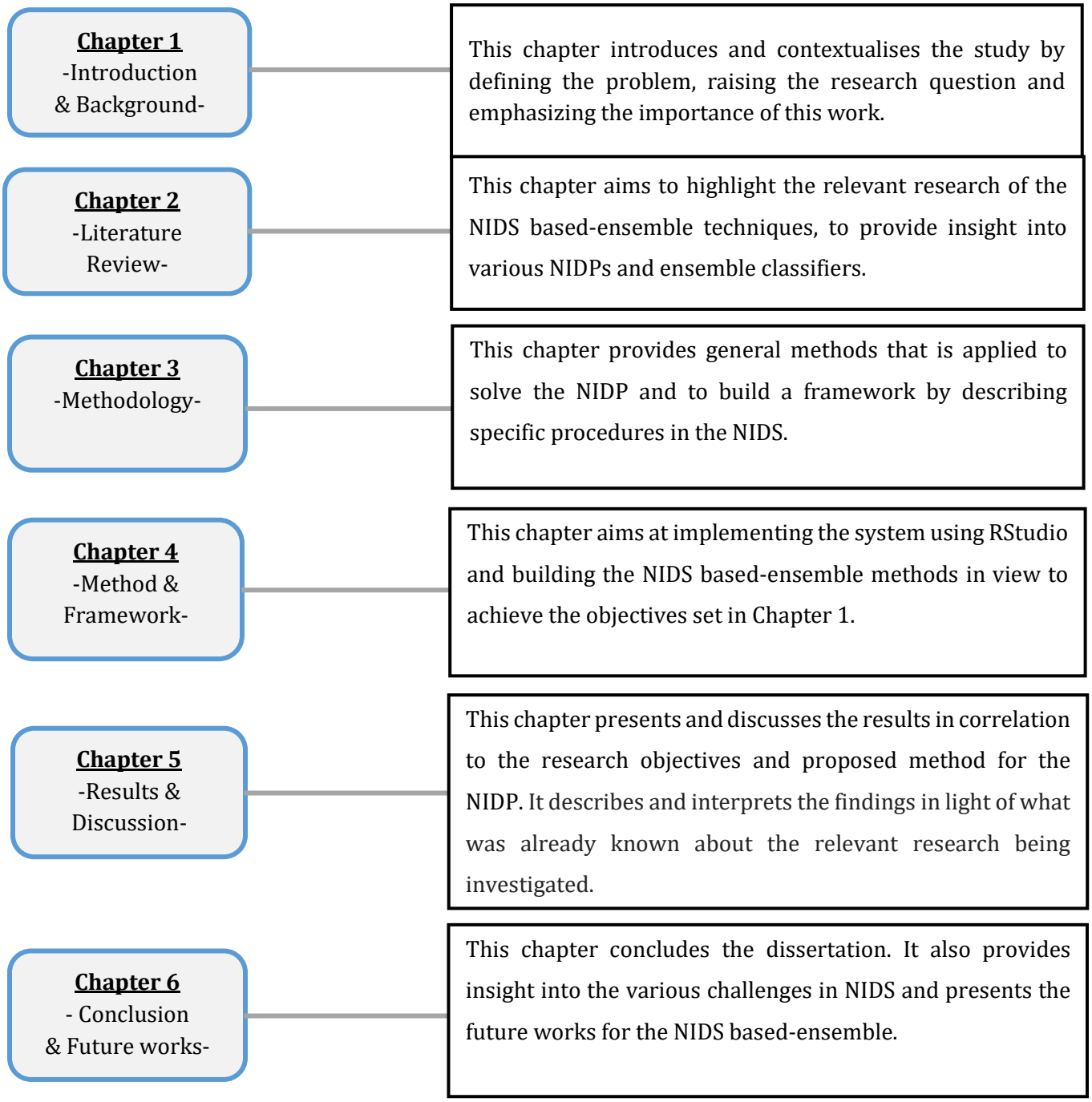
Furthermore, several works reveal that many researchers are only interested in using ML into NIDS without understanding the information that the NIDS should provide to a network administrator. This is called the "semantic gap" (Sommer and Paxson, 2010). Essentially, a NIDS must provide the relevant information about an attack occurring on the network or ongoing to the system administrator. It is therefore useless and unecessary to activate the NIDS in case the information is not provided in full. Thus, the information must be correlated with the particular setting of the network monitored by the NIDS. Traffic diversity is another factor likely to mislead predictors built with some ML techniques. Ensemble learning can overcome these difficulties, but only if it is applied with care and if researchers in the field of security work in synergy with ML specialists (Sommer and Paxson, 2010).

Particularly, ensemble learning for the NIDP will yield various solutions. This work introduces an effective methodology to mitigate various problems by analysing, testing, and creating rules for the NIDP. Other important contributions to the research are:

- Detecting attacks, unseen features and other security breaches not prevented by a single "ML" technique; new attacks with unknown instances.
- Enabling to protect the NIDS from the threats that arise with increasing network connectivity and the interdependence of information systems by providing useful information about ongoing intrusions and relevant solutions.

1.4 Dissertation layout

The figure below summaries the key chapters of the dissertation:



1.5 Summary

The purpose of this chapter was to introduce and contextualise the study by defining the problem, raising the research question and emphasizing the importance of this work. The next chapter (chapter 2), highlights relevant research of the NIDS based-ensemble techniques.

Chapter 2-Literature Review

Science knows no country, because knowledge belongs to humanity, and is the torch which illuminates the world…

-Louis Pasteur-

ML classifiers are used in network systems, IDSs, information processing such as packaging data, machine inference, etc. Hence this chapter aims to provide an insight into the IDS, the relevant research of the NIDS based-ensemble techniques and the NIDP.

2.1 Intrusion Detection System

The IDS can be classified according to certain criteria, such as the source of information, the type of analysis, the type of response and detection time.

2.1.1 Information sources

Information sources are divided into (Maiwald, 2001):

- Network-based intrusion detection system (NIDS): NIDS examines data interchanged between computers. The data is mostly gathered from the generic network flow through network segments. NIDSs gather data in the network layer, transparently to the other hosts (Albag, 2001). Particularly, the NIDS analyses network packets, decides the purpose of the traffics and detects malicious activities (Kumar et al., 2013).
- Host-based intrusion detection system (HIDS): HIDS examines data held on individual computers that serve as hosts. Data is gathered from the records of various host activities, including audit record of operation system, system logs, application program information, and so on (Thanasekaran, 2011).
- Application-based intrusion detection system (AIDS): AIDS analyses and manages the interconnection between users and systems. AIDS collects data sources from running applications as its input; it reports various replicated features and highlight features of utmost importance (Albag, 2001).

2.1.2 Analysis type

There are two types to analysing events that detect anomalous behavior:

- Misuse Detection (signature-based intrusion detection system): misuse is based on detecting attacks by examining specific patterns, such as byte sequences in the network stream or intrusive instructions used by malicious software (Lokesak , 2008). In misuse detection, the attack instances are recorded in the database and each packet of the network is compared to the attack instances in order to identify anomalous behavior. This type of intrusion is known by its performance to detect known attacks (Pharate et al., 2015).
- Anomaly Detection System (anomaly-based intrusion detection system): anomaly detection is based on detecting unusual behavior within a particular network. Attacks are therefore assumed to be different from normal activity. Profiles representing the normal behavior of users and host connections are constructed by anomaly detectors (Stewart et al., 2008).

2.1.3 Response

Responses can be grouped into:

- The passive response: IDSs forward reports to other network hosts who act appropriately on a particular matter.
- The active response**:** IDSs automatically reply to such matter or attacks (Maiwald, 2001).

2.1.4 Detection time

Two groups can be highlighted, those that detect intrusions in real time (on-line) and those that process audit data with some delay (off-line).

- Real-time: Real-time IDS works online. This IDS captures network frames in live to detect malicious actions. Real-time IDSs depend on the number of features selected; they compare these features with the functionality of incoming packets at a very high rate. Mostly, the number of features affects the resource consumption of the real-time system (Sangkatsanee et al., 2009).
- Off-line: Off-line IDS works offline. This IDS processes data sets such as KDD cup 99 data set (Pharate et al., 2015). Off-line IDS gives details about the intrusion and helps to repair the damage caused by the attack. This detection system helps to understand the attack mechanism and reduce the possibility of future attacks (Pharate et al., 2015).

The type of systems that combine both types of detection (real-time and off-line) is called hybrid (Sweeney et al., 2003).

In addition, the way of detecting an intrusion is to identify certain abnormal behaviors. The type of identification that quantifies the normal behavior of a user. Overall, three types of attacks should be taken into account (Fernandez and Porres, 2008):

1. External penetration: the attack originates from outside the network.
2. Internal penetration: an unauthorized user attacks from inside the network.
3. Resources abuse: authorized users use data or resources to which they have access, in unintended and unwanted ways (denying access to one or more legitimate users, etc.).

External attack characterizes those who do not have access to the system. Whereas internal attack characterizes those with access permission and who wish to carry out illicit activities (Li, 2004). Typically, the normal IDS is based on the source of information, the type of analysis, the type of response and detection time, as detailed in table 1.

Table 1 IDS classification (Fernandez and Porres, 2008)

	IDS		
Information source	**Type analysis**	**Response**	**Detection time**
Network based	Signature based	Active	Real-time
Host based	Anomaly based	Passive	Off-line

In particular, this work focuses on the Network based-IDS. Figure 3 illustrates the NIDS architecture:

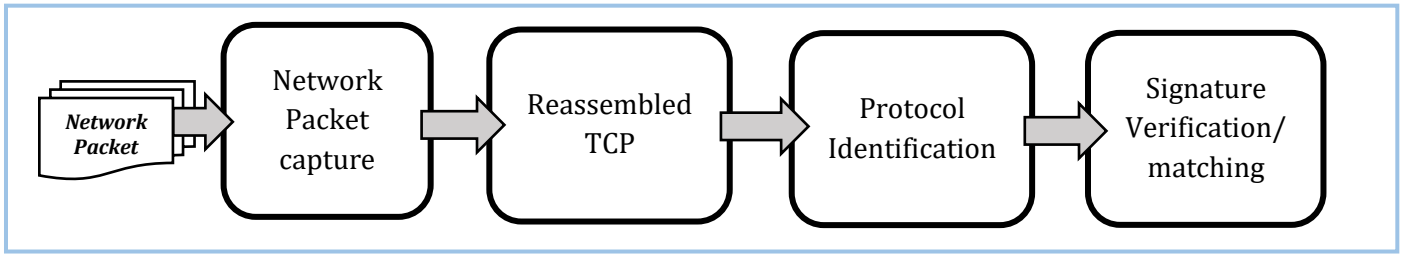


Figure 3 NIDS Architecture

The network packet or stream goes through a packet capture or a sniffer whereby packets are analysed and saved temporarily. The analysed and fragmented packets are then reassembled via the TCP (Transmission Control Protocol)-hosts. The protocol verifies the IP addresses and analyses the behavior (normal behavior or intrusion) of hosts via an algorithm for signature verification.

Furthermore, the IDS can also be represented as hybrid based-IDS (HIDS). Figure 4 illustrates the HIDS based on the:

- Approach or analyse type (anomaly and/or signature)
- Protected System (host based and/or network based)
- Structure (centralised and distributed)
- Behavior (active and passive)

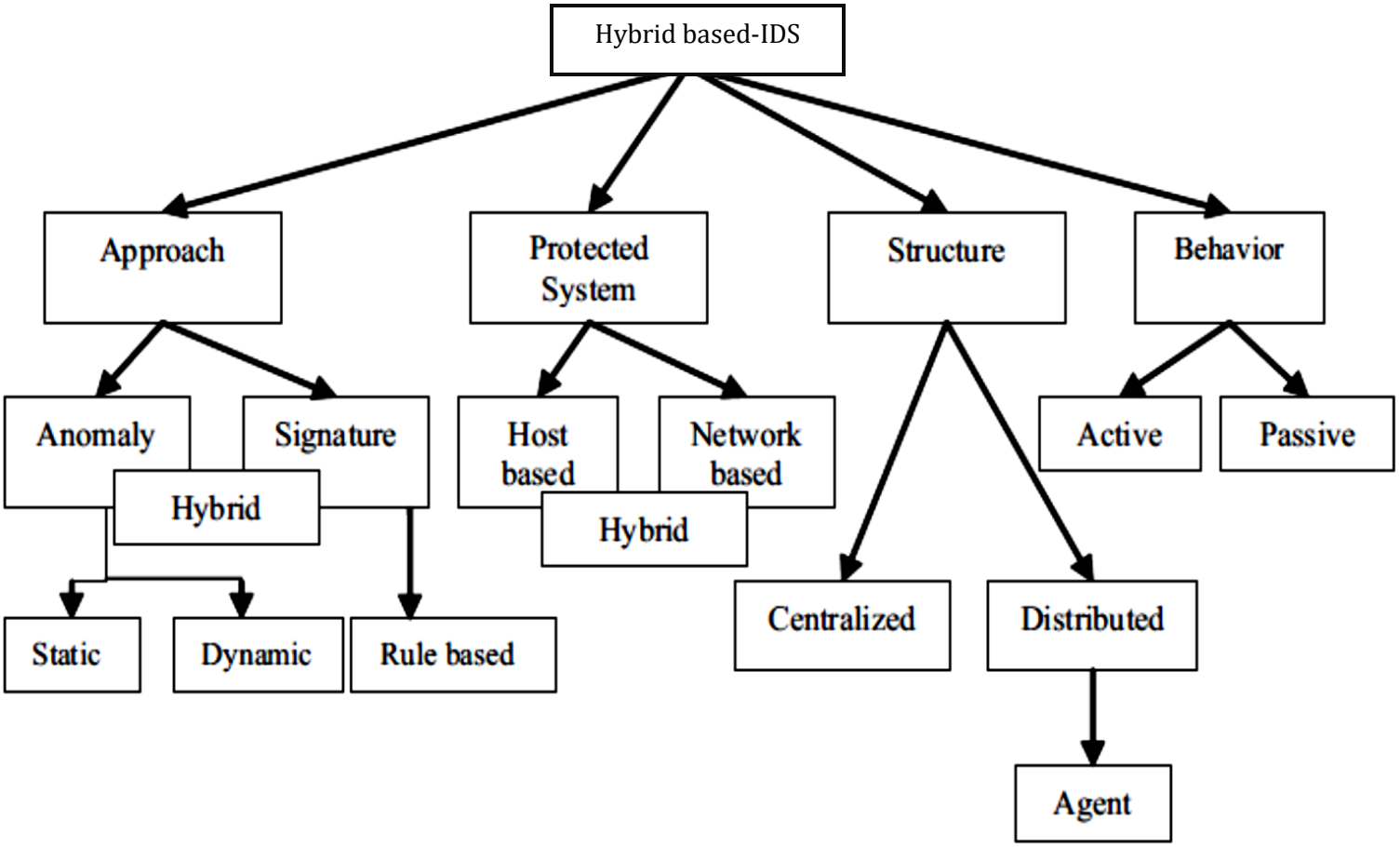


Figure 4 Hybrid based-IDS (Shanmugam and Idris , 2011).

The HIDS combines the advantages of two or more models based on some rules and algorithms. The HIDS system achieves a high degree of performance and can detect new attacks by using unsupervised and supervised methods (Prathibha et al.,2014). Specifically, this system allows to integrate the advantages of anomaly and signature based-IDS; it also merges the host based-IDS and network based-IDS. Hence the malicious activities which cannot be identified by a specific model, will then be loaded into another model for the second detection (Yu et al., 2016).

2.2 Related work

Although the list of papers on NIDS is not exhaustive, this section gives an overview of the researches by summarizing the main documentations in NIDS based-ensemble techniques. The first paper about IDSs was written by James P. Anderson (1980). Anderson (1980) was focused on the collection of records that showed abnormal use of the system, such as use outside of time, abnormal frequency of use, abnormal patterns of reference to programs or data. He also highlighted the problem of the legitimate users who often has access to confidential data; those who wish to carry out illicit activities within the system (Li, 2004). It is therefore difficult to record a feasible pathway or trail to identify misuse. His article was the first to introduce the notion of intrusion detection of the host and the IDS in general (Anderson, 1980). Seven years later, Dorothy Denning (1987) laid the foundations of IDSs development (Denning, 1987). NIDSs were introduced in 1990 by Todd Heberlein et al., (1990). In the late 90s, Artificial Intelligence (AI) researchers began to incorporate their techniques to improve the NIDS. The article was based on a detailed state-of-the-art technique according to various algorithms applied by most researchers. Particularly, ensemble techniques gave better performance than any single technique.

Borji (2007) proposed a framework on NIDS combining heterogeneous classifiers. The author introduced a combined classification approach and methodology. Artificial Neural Network (ANN), K-Nearest Neighbors algorithm (K-NN), Decision Tree (DT) and Support Vector Machine (SVM) classifiers were combined and tested on DARPA98 datasets. Domain Rating (DR) and False Acceptance Rate (FAR) metrics were illustrated to assess the performance of their approach. Outputs of four classifiers (ANN, SVM, K-NN, and DT) are merged using three combination strategies: majority vote, bayesian mean, and belief measure. The results corroborate the elasticity of the illustrated approach compared to simple classifiers on the problem of intrusion detection (Borji, 2007). Similarly, Wun-HwaChen (2005) illustrated a paper on the feasibility of introducing ANN and SVM to predict attacks based on frequency-based coding techniques. The purpose of using ANN and SVM for attack identification was to develop a generalization capability from restrained training data. In addition to diverging ANN and SVM performances, other coding techniques to predict attacks were introduced. The test bed used DARPA98 data from MIT's Lincoln Labs (Chen et al., 2005). As a result, the SVM with *tf* × *idf* (term frequency × inverse document frequency) encoding technique showed better results than the simple frequency-based technique, which in turn outperformed the ANN with that of *tf* × *idf* encoding and the ANN with a simple technique based on the frequency.

A sequence of paired *t*-tests were performed to gauge the statistical difference between these methods and coding techniques. This further confirmed that the SVM is better than the ANN for IDS and that the *tf* × *idf* encoding technique performed better than the simple frequency-based technique (Chen et al., 2005). The Receiver Operating Characteristic (ROC) curves for the SVM and ANN with various encoding techniques are shown together in figure 5 (Chen et al., 2005):

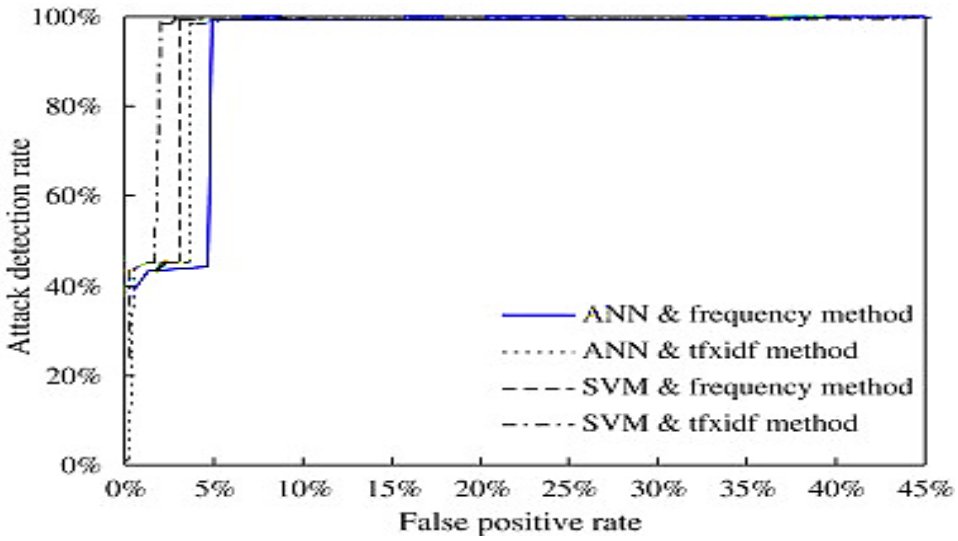


Figure 5 ROC curves for the SVM and ANN (Chen et al., 2005)

Perdisci et al. (2009) proposed an ensemble approach based on SVM for the NIDS. In this context, the payload is illustrated using the 2ν-grams technique. The value of ν is extracted according to the presentation of the payload in different functions. It should be noted that each function obtained in this way has the dimension 256^2. Then, in order to reduce the dimensionality of the data sets, the authors chose a classification algorithm based on the technique originally proposed by Dhillon et al. (2002). In the training stage, SVM based-ensemble, a framework for each data representation is constructed to model the normal instance. In the detection stage, whenever a packet is received, each representation is classified by the corresponding SVM, illustrating at output an estimate of the probability that the payload will correspond to the normal data. The outputs are then fused using a non-formable combiner, choose from the product, the average, the minimum or the maximum. The output shows an overall estimate of the probability *P* of the payload being normal. Finally, if *P* is large at a prefixed threshold, then the payload is classified as normal. DARPA98 and GATECH data sets were used for experimentations (Perdisci et al., 2009).

Additionally, Kumar and Selvaumar (2013) proposed fuzzy based ensemble classifier for the NIDS. Even using stored attack signatures, it is possible to detect Detection of Distributed Denial of Service (DDoS) attacks. Then the challenge was to identify normal traffic and DDoS. Fuzzy logic obtained solutions that cannot be obtained by another algorithm such as a neural network. The implementation of the neuro-fuzzy classifiers was gauged on the basis of the testing dataset containing non-intrusive attack types in the learning set. From the simulation experiments, the formed sets were able to detect new attacks (Kumar and Selvakumar, 2013). Then, ensemble classifiers were able to classify the new data without omitting the previously acquired knowledge.

Pre-processing was applied on the DARPA and CAIDA datasets. Pre-processing is the most important stage of any ML technique; it refers to the process of retrieving information about network traffic packets for building normalized dataset and new features. Figure 6 illustrates the block schematic of the pre-processing stage (Kumar and Selvakumar, 2013); the network traffic (input) and the normalized dataset (output).

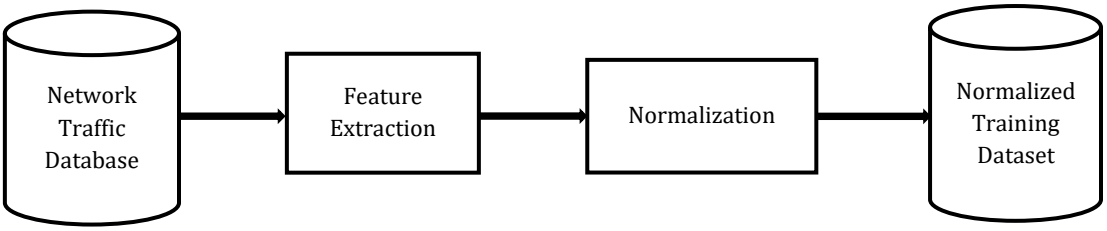


Figure 6 Pre-processing steps (Kumar and Selvakumar, 2013)

The pre-processing steps are explained as follows (Kumar and Selvakumar, 2013):

- Let 'x' be the input vector of dimension 'n', such that $x = [x_1, x_2, x_3, \ldots x_n]$. The variables x_i of the input vector are the original features.
- Let 't_x' be a vector of transformed features of dimension 't_n'.

A novel ensemble classifier for the NIDS was proposed by Jabbar et al. (2016). Different data mining techniques were used to implement the NIDS. The authors proposed ensemble NIDS using an Alternative Decision Tree (ADTree) and naïve bayes to enhance the accuracy of the detection system. NSL-KDD data set was used for experimental analysis. Discretization and Interquartile range (IQR) techniques were used for pre-processing. The results showed that the illustrated ensemble classifier outperformed single classifier in terms of accuracy (Jabbar et al., 2016).

Jabbar et al. (2017) proposed cluster-based ensemble classifier for the NIDS. ADTree and K-NN classifiers were combined to identify intrusions and K-Means clustering algorithm was applied on the Gure data set which was used for simulation. DR, FAR, Hubers Index, Rand index, and accuracy metrics were used to evaluate the method (Jabbar et al., 2017).

Afterwards, Kevric et al. (2017) developed a combinatorial classifier based on tree algorithms for the NIDS. KDDCUP'99 dataset was used for the experimentations. It was necessary to classify if the incoming network traffic was normal or as an attack, based on 41 characteristics illustrating each configuration of network traffic. 89.24% was obtained as detection precision by fusing the random tree and NBTree (Naive Bayes Tree) algorithms based on the sum rule system, thus surpassing the individual random tree algorithm. The result showed the highest performance obtained by using the complete NSL-KDD dataset (Kevric et al., 2017).

Table 2 and 3 show how combining classifier approach based on the sum rule yielded better results than individual classifiers:

Table 2 Detection performance of individual classifiers on KDDTest+ (Kevric et al., 2017)

Individual classifiers	Evaluation criteria		
	Sensitivity (%)	Specificity (%)	Accuracy (%)
Random tree	82.6	96.2	88.46
C4.5	78.0	97.2	86.29
NBTree	75.0	91.3	82.02

Table 3 Detection performance of combining classifiers models on KDDTest+ (Kevric et al., 2017)

Combining classifiers	Evaluation criteria		
	Sensitivity (%)	Specificity (%)	Accuracy (%)
Random tree + NBTree	83.9	96.2	89.24
Random tree + C4.5	83.0	96.5	88.81
NBTree + C4.5	76.7	95.6	84.98
All three	79.8	96.3	86.95

Table 4 summarizes the works on NIDS based- ensemble in the year 2017:

Table 4 Summary of earlier work on NIDS based-ensemble (Jabbar and Aluvalu, 2017)

S.No	Author	Method	Data set used	Metrics used
1	Jabbar	ADTree and Naïve bayes	NSL-KDD	DR, FAR, Accuracy
2	Kumar and selvakumar	Fuzzy based	DARPA and CAIDA	Accuracy
3	Borji Ali	ANN,K-NN,DT and SVM	DARPA98	DR and FAR
4	Perdisci	SVM based	DARPA and GATECH	Accuracy
5	Jabbar	ADTree and K-NN	GURE Data set	DR, FAR, Hubers Index, Rand index and Accuracy

Papamartzivanos et al. (2018) introduced a new approach that combines the benefits of DTs and GAs (Genetic Algorithms) with the ultimate goal of aligning legible and linguistically interpretable intrusion detection rules. The new approach allows unbalanced datasets to be processed and intrusion detection rules to be generated allowing an NIDS to detect common and rare types of attacks. DT and GA have been combined to find the optimal solution to generate intrusion detection rules and respond to two qualities in the field, namely: the interpretability of detection rules to increase human understanding and the generation of precise detection rules (Papamartzivanos et al., 2018).

A weighted selection probability function was provided in regard to the GA for enhancing DTs that are not biased toward neglecting classes represented by a lower percentage of records in the dataset. Then the approach was designed against other state-of-the-art techniques and ML in an extensive test-bed containing three datasets, namely KDDCup'99, NSL-KDD and UNSW-NB15. Six various classification measures have been used to evaluate and compare the methodology. And the results show that the proposed approach outperformed equivalent methods in terms of Mean F-Measure (MFM), Average Accuracy (AvgAcc) and Attack Accuracy (AttAcc) classification metrics, while reducing false alarms (Papamartzivanos et al., 2018).

Additionally, Ahmad et al. (2018) presented a draft and a method for the comparison of SVM, Random Forest (RF) and Extreme Learning Machine (ELM) for the NIDS. This method analyses and compares the techniques needed to overcome the limitations of using IDS classification in huge data sets, such as system and unbalanced network traffic. SVM, SVM Radial Basis Function (RBF), RF and ELM were used as main classifiers. These algorithms are very efficient because of their classification capacity. Figure 7 illustrates the proposed model for the IDS (Ahmad et al., 2018):

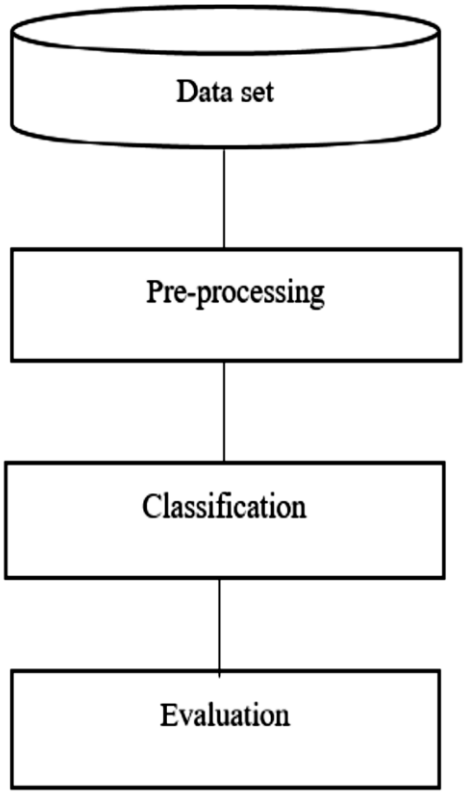


Figure 7 Proposed model of the NIDS (Ahmad et al., 2018)

As previously illustrated, the initial objective of this article was to investigate the performance of the four classifiers (SVM, SVM (RBF)**,** RF, and ELM) in IDS. The key stages of the proposed model include (Ahmad et al., 2018):

- The dataset (the NSL-knowledge discovery and data mining dataset is used, which is considered as a benchmark in the evaluation of IDS and comparison of the results)
- Pre-processing (process raw data into machine-readable form)
- Classification (SVM, SVM (RBF), RF and ELM)
- Result evaluation.

As a result, the comparison performances of SVM (Linear), SVM (RBF), RF, and ELM on 20% testing and 80% training data samples is shown in Figure 8. ELM performance is better than SVM (Linear), SVM (RBF) and RF for full data samples, whereas SVM (RBF) shows improved accuracy over RF and ELM for half data samples. SVM (Linear) surpassed the other techniques on 1/4 of the data samples, as illustrated in Figure 8 (Ahmad et al., 2018):

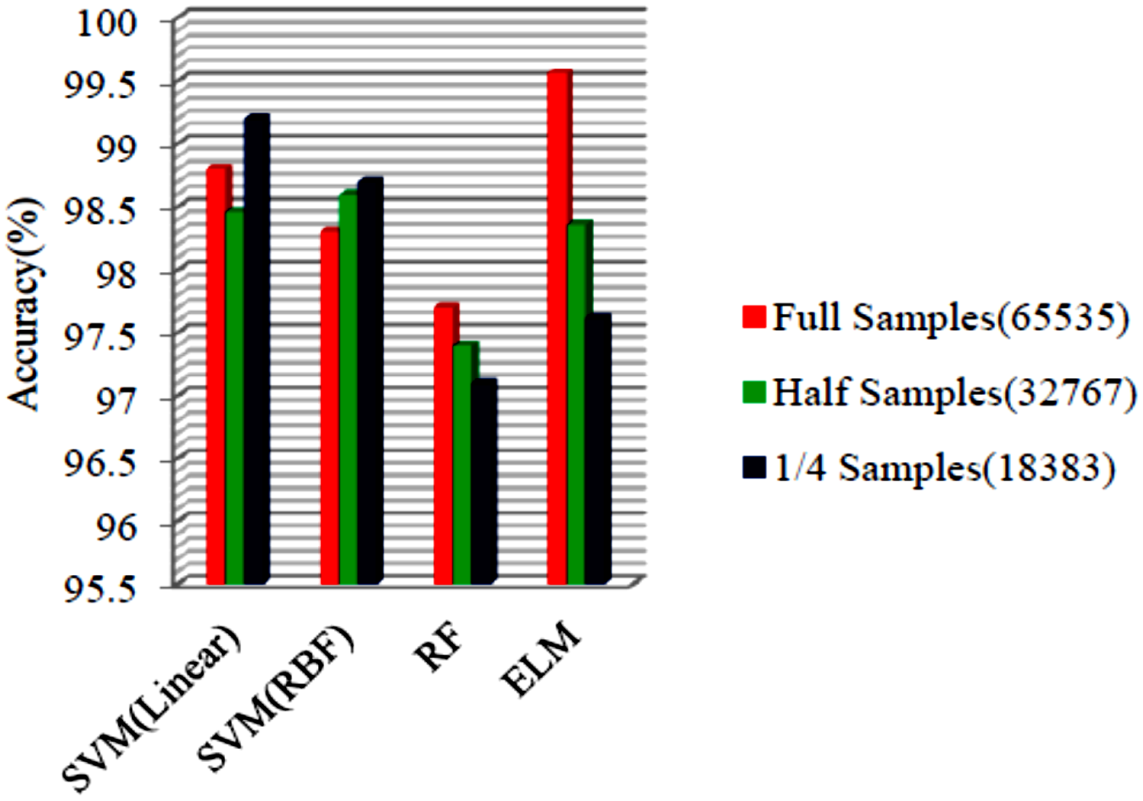


Figure 8 Accuracy of SVMs, RF, and ELM (80% training and 20% testing) (Ahmad et al., 2018).

Finally, the authors corroborate that ELM was an appropriate technique for IDS designed to analyse a considerable amount of data. ELM surpasses other techniques (SVMs and RF) in accuracy, recall and precision on the complete data samples comprising 65,535 records of features containing normal and intrusive instances. ELM can be applied and used to solve several clustering, classification, regression and feature engineering problems; its performance is similar to SVM or other ML techniques. Hence, authors intent to further investigate ELM technique and its performance in feature selection and classification techniques (Ahmad et al., 2018).

2.3 Network Intrusion Detection Problem

The NIDP is divided into four classes: probes, denial of service, user to root and remote to local (Mukkamala and Sung, 2003).

2.3.1 Probes

Probes are actions taken by an attacker in gathering information on a specific machine within a network (Kendall, 1999). Without probes, an intruder has a hard time finding the vulnerabilities on his target. That is why it is essential to detect probes. Many programs and protocols have been implemented to analyse a network. The most reputed is probably "nmap" (Network Mapper), a tool that is able to identify active machines and active ports in a network. The information provided by this protocol is important because knowing that port 22 is active, for instance, implies that a remote machine or server is potentially vulnerable or misconfigured. If port 22 is open, the intruder may conclude that the machine is exposing its contents without being encrypted. "Nmap" is not only restricted to identifying active ports; it is also able to detect the type and version of the server or the type and version of the OS (Kendall, 1999).

Other attacks such as "saint" and "satan" are reputed in the discovery of failures and vulnerabilities in a specific system. These scanning tools allow unskilled hackers to automatically find vulnerabilities on a huge number of machines. A typical attack scenario would involve a first stage where the intruder tries to analyse the network that he wants to compromise. Through probe or scan, the attacker will have a complete map of the active machines and services running on the network. Table 5 depicts the actions taken by an attacker during probe.

Table 5 Type of probe attack (Kendall, 1999)

Specific Type	Description
Probe (Machines)	Indicate the types and numbers of machines on network
Probe (Services)	Define the services supported by a particular system
Probe (Users)	Define the names or relevant information about users having accounts on a specific system

Table 6, which was adapted by Kendall (1999) illustrates various types of probes with certain attributes for a particular type such as services used by the attack, the platforms vulnerable to this kind of probe, the type of mechanism used by the attack, the time required to implement it and the effect caused by the attack (Kendall, 1999).

Table 6 Probe attacks (Kendall, 1999)

Name	Service	Vulnerable Platforms	Mechanism	Time to implement	Effect
Ipsweep	Icmp	All	Abuse of feature	Short	Find active machines
Mscan	Many	All	Abuse of feature	Short	Looks for known vulnerabilities
Nmap	Many	All	Abuse of feature	Short	Find active ports on a machine
Saint	Many	All	Abuse of feature	Short	Looks for known vulnerabilities
Satan	Many	All	Abuse of feature	Short	Looks for known vulnerabilities

2.3.2 Denial of Service

In a denial of service (DoS), resources within a network become inaccessible to legitimate users because of the intrusion. The resource can be network bandwidth, computer memory or computing power. The Address Resolution Protocol (ARP) is a protocol used to convert network layer address; for example the IP address, the Media Access Control (MAC) address into the link layer address. Each computer on a network is provided with an ARP table that traces network addresses to link layer addresses of other machines or network devices (Kendall, 1999).

In an "ARP poisoning" attack, an attacker sends ARP access denied to a user on the same network or to an ARP request by intercepting the request's destination and muddling the information contained in the victim's ARP table. In this case, it is feasible for an attacker to refute access to a resource to one or more users within a network. For instance, based on the network pattern, it is conceivable for an attacker to edit the entry related to a gateway in the victim's ARP tables on the network.

In this case, the victim may no longer be able to access the Internet. "ARP poisoning" is not illustrated in the KDD99 dataset, but the concept of DoS can relatively be grasped from this attack (Kendall, 1999). Table 7 shows various types of DoS with certain attributes targeting a particular type services, the platforms vulnerable to this type of DoS, the type of mechanism used by the attack, the time required to implement it and the effect caused by the attack. This DoS list is not exhaustive, but provides an overview of the variety of this type of attack (Kendall, 1999).

Table 7 DoS attacks (*) [1] (Kendall, 1999)

Name	Service	Vulnerable Platforms	Mechanism	Time to implement	Effect
Apache2	http	Any apache	Abuse	Short	Crash httpd
Back	http type	Any apache	Abuse/Bug	Short	Slow server response
Land	N/A	SunOS	Bug	Short	Freeze machine
Mailomb	smtp	All	Abuse	Short	Annoyance
SYN Flood (Neptune)	Any TCP	All	Abuse	Short	Deny service on one or more ports for minutes
Ping of Death	icmp	None(*)	Bug	Short	Crash the targeted computer
Process Table	Any TCP	All	Abuse	Moderate	Deny new processes
Smurf	icmp	All	Abuse	Moderate or Long	Network Slowdown
Syslogd	syslog	Solaris	Bug	Short	Kill Syslogd
Teardrop	N/A	Linux	Bug	Short	Reboot machine
Udpstorm	echo/chargen	All	Abuse	Short	Network Slowdown
ARP Poisining	ARP	All	Abuse	Short	Deny access to one or more machines on the network

2.3.3 User to Root

In a root user attack (U2R), an attacker logs on to a computer as a legitimate user with unrestricted privileges. By exploiting some vulnerability of the program installed on the computer, the user can easily access the confidential information. The purpose of this class of exploits is primarily to enter confidential information and administrator rights on the computer attacked in order to obtain full control.

[1] (*) the bug has been fixed on most platforms since 1998 (Kendall, 1999)

There are several types of U2R attacks. The buffer overflow is certainly the major vulnerability used by hackers when attempting to obtain privileged access to a computer (Cowan et al., 2000). The goal of an attack by buffer overflow is to corrupt a program launched with access uncontrolled to take control of it. If the program holds root privileges, the intruder can simultaneously execute a command to get to the root shell. In this case, the attacker completely controls the host computer running the vulnerable program. The attack is carried out in two stages:

1. The hacker eventually finds a way to get the correct code to run a root shell in the program's memory. To handle this, the attacker uses a buffer with an identification of non-existent or poorly conducted limits.
2. The attacker tries to corrupt the pointer of the stack to click it to its malicious code.

Several options are possible, but the best known is to overwrite a function's return address to point to the first instruction of the attacker's code. This attack is also referred to as "stack smashing attack" (Aleph, 1996). Table 8 shows various types of U2R with certain attributes for a particular type such as service used by the attack , the platforms vulnerable to this type of U2R, the type of mechanism used by the attack, the time required to implement it and the effect caused by the attack (Kendall, 1999).

Table 8 U2R attacks (Kendall, 1999)

Name	Service	Vulnerable Platforms	Mechanism	Time to implement	Effect
Eject	Any user session	Solaris	Buffer overflow	Medium	Root shell
Ffbconfig	Any user session	Solaris	Buffer overflow	Medium	Root shell
Fdformat	Any user session	Solaris	Buffer overflow	Medium	Root shell
Loadmodule	Any user session	SunOS	Poor environment sanitation	Short	Root shell
Perl	Any user session	Linux	Poor environment sanitation	Short	Root shell
Ps	Any user session	Solaris	Poor temp file management	Short	Root shell
Xterm	Any user session	Linux	Buffer overflow	Short	Root shell

2.3.4 Remote to Local

In a Remote to Local (R2L), the attacker logs in from a computer located outside the targeted network and exploits the vulnerability to access a computer on the LAN (Local Area Network). A prerequisite is that the attacker can send network packets to the victim host. Instances of R2L attacks include “warezmaster” and “warezclient”. Both attacks exploit weaknesses of the file transfer protocol (FTP) (Kendall, 1999):

- Warezmaster: grants any user access to write on the FTP server. An intruder can use this bug to upload unauthorized files on the server.
- Warezclient: involves a user downloading the illegal files from the hidden directory generated during the warezmaster attack.

Other R2L attacks such as "imap" (Internet Message Access Protocol), "named" and "sendmail" use bugs in reputed Internet protocols such as SMTP (Simple Mail Transfer Protocol) and DNS (Domain Name System). Attacks using configuration errors in the system include "dictionary", "ftp-write", "guest" and "Xsnoop". Keeping the system up to date remains the best solution against R2L attacks. Table 9, shows various types of R2L attacks with certain properties for a particular type such as service used by the attack, the platforms vulnerable to this type of R2L, the type of mechanism used by the attack, the time required to implement it and the effect caused by the attack.

Table 9 R2L attacks (Kendall, 1999)

Name	Service	Vulnerable Platforms	Mechanism	Time to implement	Effect
Dictionary	telnet,rlogin,pop, imap,ftp	All	Abuse of feature	Medium	User-level access
Ftp-write	ftp	All	Misconfiguration	Short	User-level access
Guest	Telnt,rlogin	All	Misconfiguration	Short	User-level access
Imap	Imap	Linux	Bug	Short	Root shell
Named	Dns	Linux	Bug	Short	Root shell
Phf	http	All	Bug	Short	Execute commands as user http
Sendmail	Smtp	Linux	Bug	Short	Execute commands as root
Xlock	X	All	Misconfiguration	Medium	Spoof user to obtain password
Xsnoop	X	All	Misconfiguration	Short	Monitor keystrokes remotely

2.4 Machine Learning

First of all, understanding Machine learning (ML) before ensemble learning is of paramount importance. ML refers to the automated detection of significant patterns in the data (Shalev and Ben, 2014). In general, ML tools are intended to equip programs with the ability to learn and adapt. Learning is the process of converting experience into expertise or knowledge. The training data, which represents the experience, is fed by a learning algorithm. The output or result is an expertise, which usually takes the form of another computer program capable of performing a task (Shalev and Ben, 2014). Specifically, ML can be thought of as a branch of AI that uses statistics extensively. As part of the basic statistical learning, the learner has access to the domain set, the tag set and the training data. ML's ultimate goal is to use the unique strengths and capabilities of computers to enable human intelligence to perform tasks that go beyond human abilities (Shalev and Ben, 2014).

ML algorithms can be divided into three types (Shalev and Ben, 2014):

1. Supervised learning:

Involves obtaining a training dataset in which each entry is tagged. A tag set indicates which class it belongs to (normal or attack). In supervised learning, the algorithm constructs a model that should be able to separate the examples with other labels. There exist many supervised learning algorithms such as ANN, DT, SVM, etc. The aim is to learn a mapping from x to y, from a training set of pairs (x_i, y_i). The $y_i \in$ Y are called the labels of instances x_i. If the labels are numbers, y = $(y_i)i^T \in [n]$ denotes the column vector of labels. The most frequent of supervised learning disadvantage is the need for a set of labelled data (Chapelle et al., 2009). Supervised learning has the advantage of being able to classify similar instances. However, this accuracy drops significantly when the new examples are not so similar to the ones in the training set (Laskov et al., 2005).

2. Unsupervised learning:

Do not require labelled dataset. The most used technique of unsupervised learning is called clustering. In this case, the algorithm exploits the similarity of the examples in order to form clusters or groups of instances. Instances belonging to the same cluster are expected to share similar attributes and belong to the same class. Let X= $(x_1,\ldots, x_n)$ be a set of n instances or points, where $x_i \in$ X for all $I \in [n]$:= {1,…,n}. Assuming that the points are drawn independently and identically distributed from a common distribution on X. Defining the ($n \times d$)-matrix X = $(x_i^T)i^T \in [n]$ that contains the data points as its rows. Unsupervised learning aims to find interesting structure in the data X (Chapelle et al., 2009).

Unsupervised learning presents some drawbacks; the manual choice of the number of cluster that the algorithm must form, the low accuracy of the prediction and the fact that the meaning of each cluster must be interpreted to understand the output. However, unsupervised learning is more resistant to strong variations than supervised learning. In particular, unsupervised learning is able to detect new types of attacks much better than supervised learning (Balon-Perin, 2012).

Another type of learning method that falls between the two main types is semi-supervised learning: semi-supervised uses both supervised and unsupervised learning techniques. In semi-supervised learning, the algorithm receives some supervision information. Semi-supervised is then an extension of unsupervised and supervised learning which includes additional information relevant to other learning paradigm (Triguero et al., 2015). Often, this information will be the target associated with some of the instances. Then the data X= $(x_i)_{i \in [n]}$ can be divided into two sections $X_l := (x_1,..., x_l)$, for which labels $Y_l := (y_1,..., y_l)$ are provided, and the points $X_u := (x_{l+1},..., x_{l+u})$, the labels of which are not known (Chapelle et al., 2009). Typically, Semi-supervised algorithms are suitable tools to tackle training sets with large amounts of unlabelled data and a small quantity of labelled data. Semi-supervised techniques use unlabelled samples to either modify or reprioritize the hypothesis obtained from labelled samples alone (Triguero et al., 2015).

3. Reinforcement learning:

Involves learning what to do and how to map situations to actions; to maximize a numerical reward signal. Initially, the learner is not told which actions to take, as in other types of ML, but rather to discover which actions produce the most benefits by trying them. The agent (learner) must then have information and goals relating to the state of the environment or the system. The information must include sensation, action and purpose (Sutton and Barto, 2011). Reinforcement learning maximizes a reward signal instead of relying only on known instances. Then in order to maximize rewards, the learner must choose actions tried before and which have proven effective in producing rewards. But to discover such actions, it must try unknown actions. The learner has to exploit and explore known instances in order to obtain rewards and to make better actions in the future. A reward signal defines the purpose of a reinforcement learning problem. At each time step, the environment or the system sends to the reinforcement learning agent a unique number, a reward. Specifically, the reward signal implies what are the good and bad events for the agent (Sutton and Barto, 2011).

Figure 9 illustrates a basic reinforcement learning model. Initially, a controller receives the controlled system's state and a reward associated with the last state transition. It then calculates an action which is sent back to the system. In response, the system makes a transition to a new state and the cycle is repeated (Szepesvári, 2010).

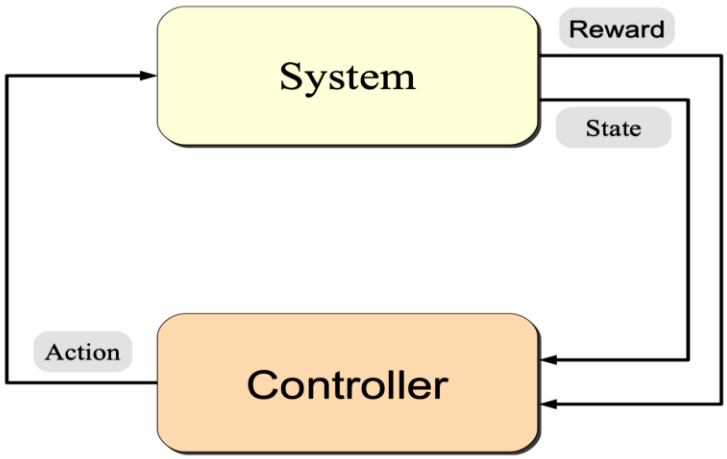


Figure 9 Reinforcement learning scenario (Szepesvári, 2010)

Before delving deeper into ML, this section introduces the Artificial Neural Network (ANN) as part of the ML models. There exist many ML models (Bayesian networks, GA, etc.). In particular, NIDS applies ANN to various types of ML techniques (supervised, unsupervised and reinforcement) in order to enhance its overall performance. An ANN is a mathematical model that tries to simulate the structure and functionalities of biological neural networks. A biological neuron includes the brain, spinal cord and peripheral ganglia. Information comes into the neuron through dendrite, soma processes the information and passes it on via axon (Andrej et al., 2011).

Particularly, building blocks of every ANN is the artificial neuron which is a simple mathematical function. This function or model has three simple sets of rules (Andrej et al., 2011):

1. Multiplication: at the entry of the artificial neuron, the inputs are weighted; each input value is multiplied by an individual weight.
2. Summation: in the middle section of the artificial neuron, all weighted inputs and bias are summed up by the sum function.
3. Activation: at the output of the artificial neuron, the sum of the previously weighted inputs and the bias passes through the activation function (transfer function).

Figure 10 illustrates the working principle of an artificial neuron:

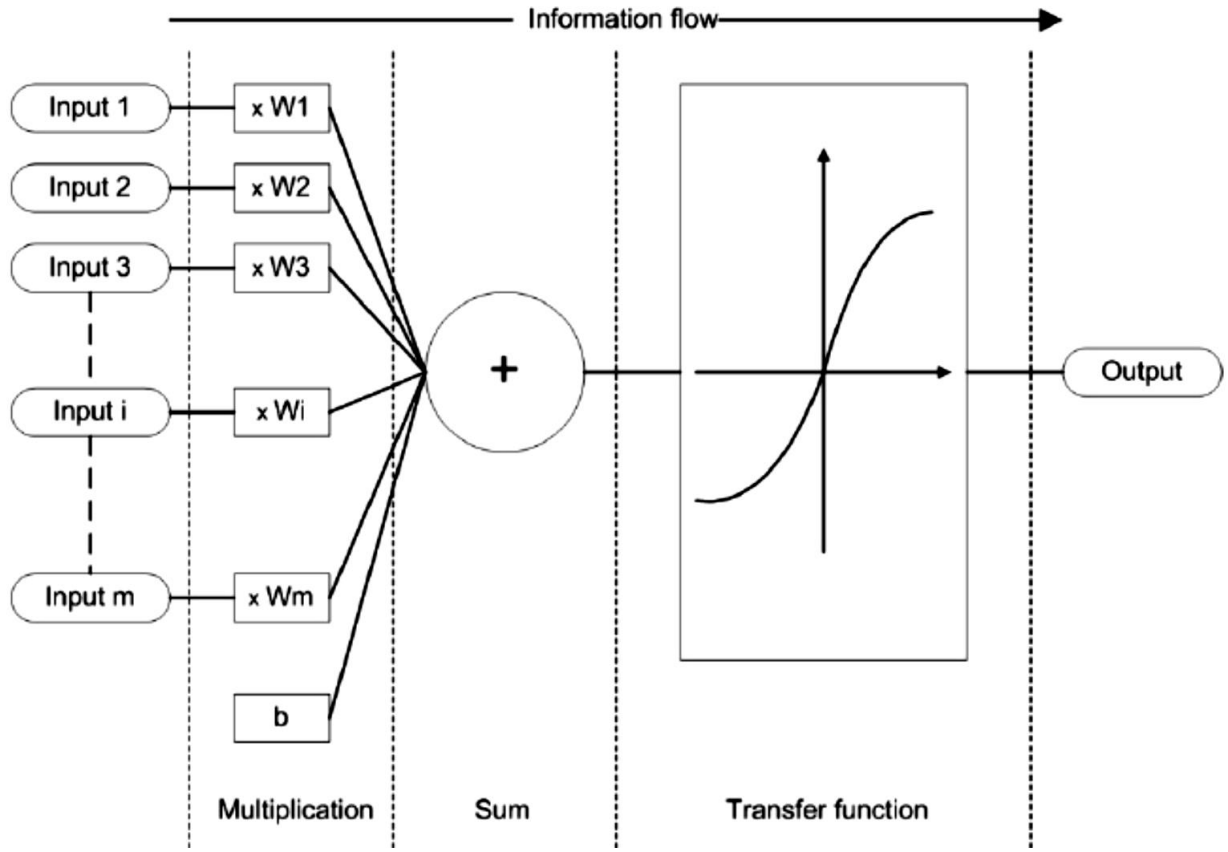


Figure 10 Working principle of an artificial neuron (Andrej et al., 2011)

The mathematical description of an artificial neuron model is as follows (Andrej et al., 2011):

$$y(k) = F\left(\sum_{i=0}^{m} w_i(k) \cdot x_i(k) + b\right) \tag{2.1}$$

Where:

- $x_i(k)$ is the input value in discrete time k where i goes from 0 to m,
- $w_i(k)$ is weight value in discrete time k where i goes from 0 to m,
- b is bias,
- F is a transfer function (step function, linear and non-linear function, etc.),
- $y_i(k)$ is output value in discrete time k.

Moreover, the way that individual artificial neurons are interconnected is called topology or graph of an ANN. ANN topologies are divided into two basic classes (Andrej et al., 2011):

- Simple feed-forward topology (acyclic graph): information flows unidirectionally from inputs to outputs.
- Simple recurrent topology (semi-cyclic graph): information flows bidirectionally; also flows in opposite direction (input-output).

Figure 11 depicts the topology of an ANN:

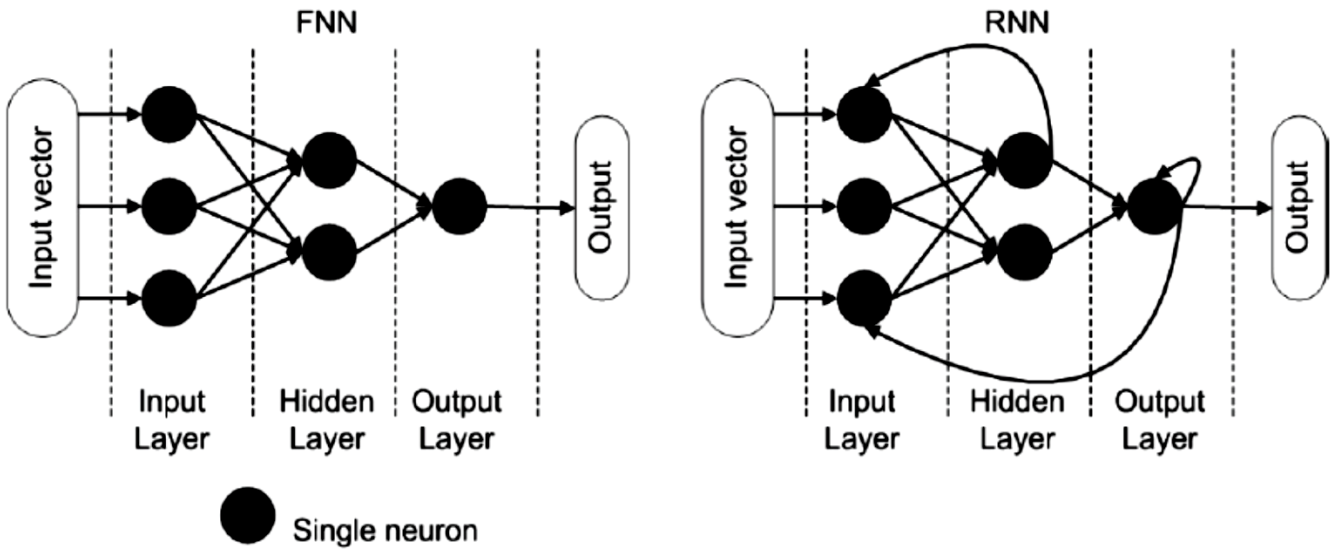


Figure 11 Feed-forward and recurrent topology of an ANN (Andrej et al., 2011)

An ANN has some advantages and disadvantages described below (Shah and Trivedi, 2012):

- It has self-learning capability.
- When an element of the neural network fails, it can continue without problem due to their parallel nature.
- A neural network learns and does not need to be reprogrammed.
- ANN requires training to operate;
- High processing time for large neural networks.
- The topology of ANN differs from the topology of microprocessors, hence needs to be emulated.

Due to the self-learning ability, an ANN can be very useful in addressing the NIDPs. In particular, the ANN allows to detect abnormal activities and reduce the false alarms in the network. There exist different types of the ANN. For instance, the Back Propagation Neural Network (BPNN), Self-Organizing Maps (SOM), RBF, Simulated Annealing neural network (SA), etc. Then to alleviate the NIDPs, researchers use a simple ANN approach or multi-layer approach of the ANN (Shah and Trivedi, 2012).

Generally, the ANN based-IDS can be classified as (Shah and Trivedi, 2012):

1. Simple ANN Based-IDS: uses one "ANN" technique; BPNN, KNN, SOM or SA.

Because of its accuracy and persistence compared to other ANN techniques, BPNN has attracted the attention of many researchers.

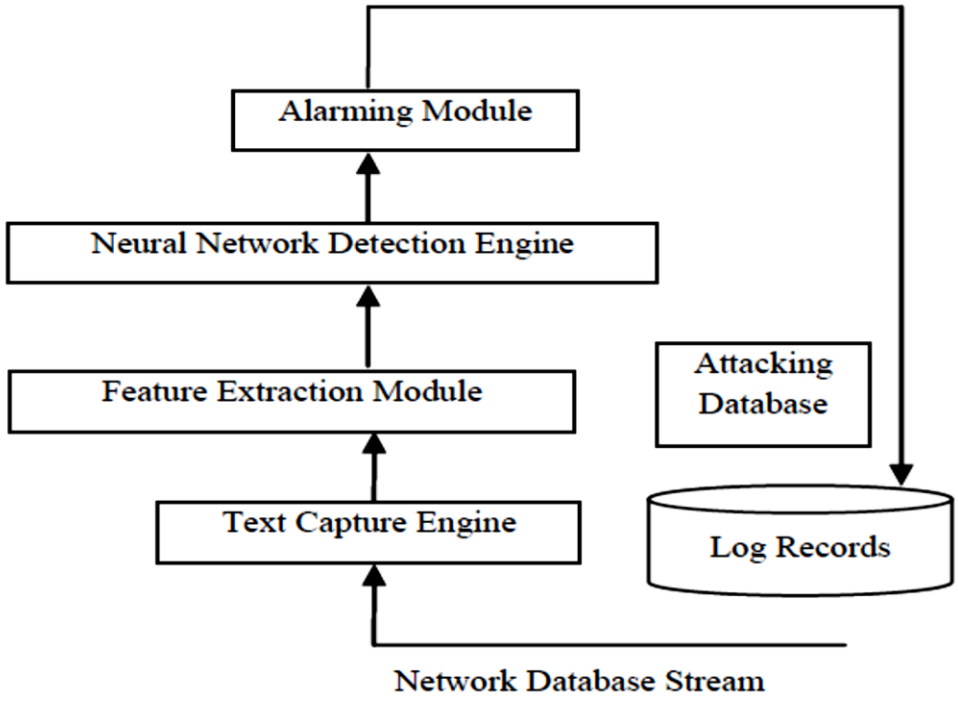


Figure 12 BPNN Model (Wei et al., 2010)

Figure 12 illustrates the BPNN model. Neural network detection engine analyses and processes the network data given by the feature extraction module. In case of an intrusion, the detection engine sends a warning message to the user, records the attack information and updates the attack database in order to relearn the neural network classification engine (Wei et al., 2010).

2. Hybrid ANN Based-IDS: uses more than one ANN technique (multi-layer approach).Hybrid ANN based-IDS aims at implementing more accurate and flexible techniques; adaptable to changes in the network system (Shah and Trivedi, 2012). Hence two or more layers approach can be implemented.

Shah and Trivedi (2012) implemented two different layers:

- First layer (Classifier Layer/KNN Layer): used to reduce false alarms; it analyses and classifies the network traffic (into DoS, Probe, R2L, U2R and normal) selected from the KDD99 dataset. KNN has been applied for separating different files (attacks and normal) containing 41 inputs and 4 outputs of the original KDD99.
- Second Layer (Decision Layer/SVM Layer): Support Vector Machine (SVM) has been used to improve the detection rate. SVM has been employed because of two reasons :
  - SVM is more efficient (Haijun et al., 2007).
  - SVM is less complex than other neural network models because of to the number of hidden layers, number of nodes for each layer and transfer functions (Shah and Trivedi, 2012).

Furthermore, some NIDS developers are changing ANN for a pattern recognition technique (Mane et al., 2013). In this case, feed-forward neural network can be used to implement pattern recognition technique[2]. During the training phase, the neural network parameters are modified to correspond to the outputs of the input patterns (each input model is selected by a vector property selected in the network connection records) (Mane et al., 2013). Some researchers are also using SOM + SA. SOM is an unsupervised algorithm that works with a nonlinear dataset; weight adjustment is determined by learning rate and the difference between the input pattern and the winner neuron's weight (Shah and Trivedi, 2012). SA reduces the training steps. Specifically, SA is looking for the optimal solution; it moves randomly in the solution space in search of a solution that minimizes the value of the objective function. In general, simple ANN is fast but not accurate, while a multilayer approach is more accurate but not fast. However, the combination of different ANN techniques can improve the overall accuracy (Shah and Trivedi, 2012).

Additionally, ML is subdivided into four techniques, namely pattern classification, single classifier, hybrid classifier and ensemble classifier.

[2] Also referred to as pattern classification (Jain et al., 2000)

2.4.1 Pattern classification

Pattern classification studies "how machines can observe the environment, learn how to distinguish patterns of interest from their background, and make reasonable decisions about the categories of the patterns (Jain et al., 2000); it is also the action of taking raw data and activity on a category of data (Michalski et al., 1998). Supervised and unsupervised learning can be used to solve different pattern classification problems. Supervised learning is based on the use of learning data to create a function, in which each of the training data contains a pair of the input and output vector. The learning task is to calculate the distance between the input–output instances to build a model. When the model is built, it can classify unknown instances into learned class tags (Chih-Fong et al., 2009).

2.4.2 Single classifier

Single classifier or learning is based on using an ML algorithm for a specific problem. For instance, k-nearest neighbor, SVM, GA, ANN, DTs, ELM, ADTree, SOM, naïves bayes networks, fuzzy logic, etc. Many researches have shown that K-NN and SVM are the most commonly used techniques of a single approach to intrusion detection problems (Chih-Fong et al., 2009).

2.4.3 Hybrid classifier

The hybrid classifier consists of combining several ML techniques to improve system performance. Particularly, hybrid classifier cascades various classifiers, such as neuro-fuzzy techniques. In order to eliminate unseen training instances from each class of attack, clustering-based approach can be employed to pre-process the input samples. Hybrid classifier methods may also be based on either supervised or unsupervised learning techniques (Chih-Fong et al., 2009). In general, the hybrid approach is to take input raw data and generate its output results; take the initial dataset and generate the final results (Jang et al., 1996).

There are three strategies to design hybrid classifiers (Chih-Fong et al., 2009):

- Cluster+ Single methods.
- Cascaded hybrid methods.
- Integrated-based hybrid methods.

Many researches show that integrated-based hybrid methods are the most considered hybrid classifier design for intrusion detection.

2.4.4 Ensemble classifier

Ensemble classifier or learning was introduced for the first time in the late 80s. Hansen and Salamon (1990) proved that the combination of several ANNs could significantly enhance the overall accuracy of the predictions (Hansen and Salamon, 1990). The term "ensemble" techniques are based on building a set of weak classifiers or learners and classifying new data points by taking a weighted sum of their predictions; several studies proved that ensemble performs better than any single classifier (Kittler et al., 1998). The objective of ensemble classifier is to create multiple classifiers with a relatively correlative bias, then combine their outputs, such as averaging to reduce the variance (Polikar, 2012).

Considering an ensemble of three classifiers $\{t_1, t_2, t_3\}$ and a new case X. If the three classifiers are identical, then when $t_1(X)$ is wrong $t_2\ (X)\ and\ t_3\ (X)$ will also be wrong. However, if the errors made by the classifiers are uncorrelated, then when $t_1(X)$ is wrong $t_2\ (X)\ and\ t_3\ (X)$ may be correct, so that a majority vote will correctly classify X. If the error rates of L hypotheses t_l are all equal to p < ½ and the errors are independent, then the probability that the majority vote will be wrong is the area under the binomial distribution where more than L/2 hypotheses are wrong. Hence it is often possible to construct very good ensemble techniques (Dietterich, 2000). Figure 13 depicts a simulated ensemble of 21 hypotheses, each having an error rate of 0.3. The Area Under the Curve (AUC) for 11 or more hypotheses being simultaneously wrong is 0.026, which is much less than the error rate of single hypotheses.

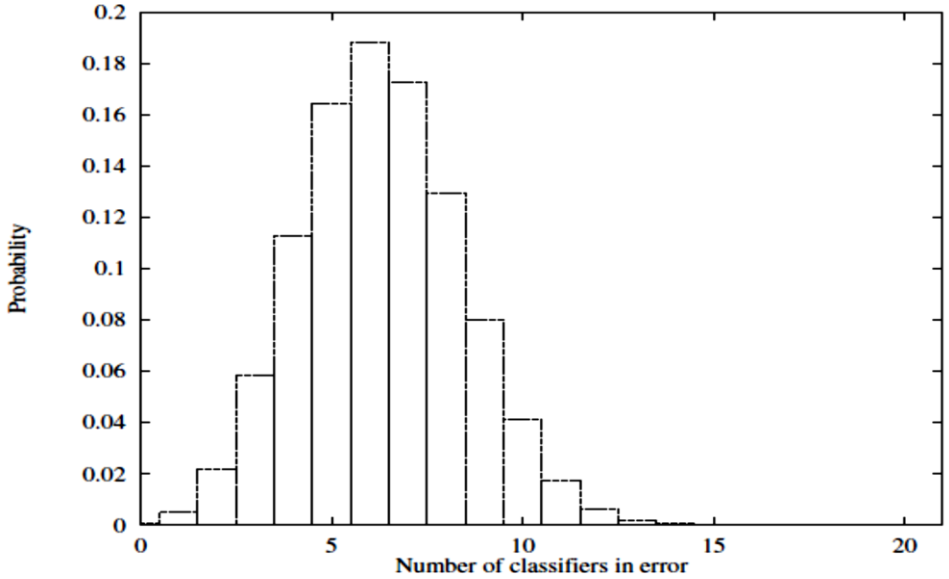


Figure 13 Simulation of 21 hypotheses based-ensemble (Dietterich, 2000)

Figure 14 illustrates the tree reasons why an ensemble classifier may work better than a single classifier (Dietterich, 2000):

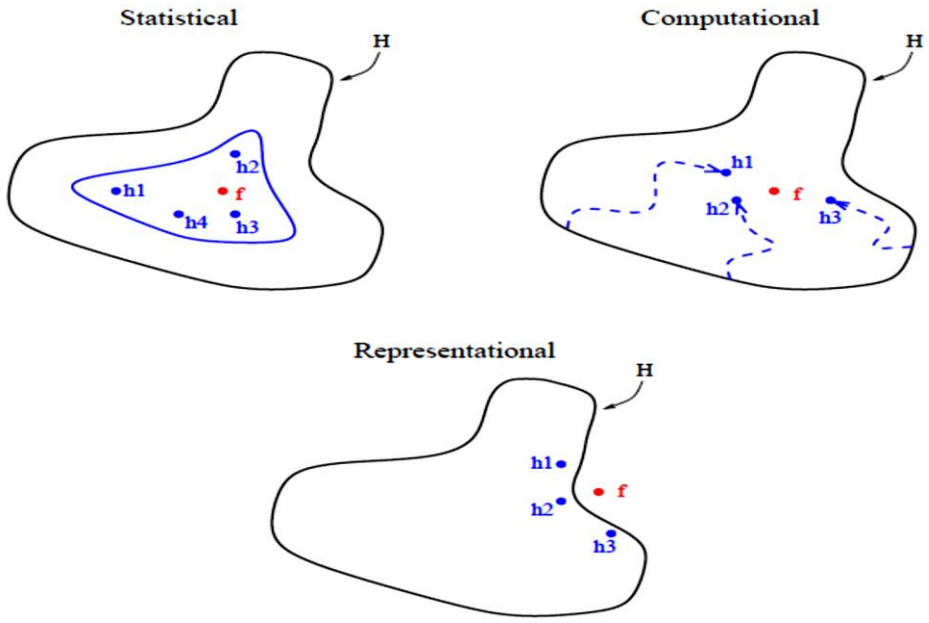


Figure 14 Three reasons why an ensemble classifier may work better than a single classifier (Dietterich, 2000).

Let $F : \mathbb{R}^n \longmapsto \Omega$, be the function that correctly assigns any pattern z to its true class ω_z, and assume to have n different classifiers $H_1, H_2, \ldots H_n$, in a hypothesis space $\mathcal{H}$, built so that they approximate F. First reason (*Statistical*): a learning algorithm can be seen as a search algorithm that attempts to find a function $H \in \mathcal{H}$ as close as possible to F. When the size of the training dataset D is small compared to the size of $\mathcal{H}$, the algorithm may find many different functions $\{H_i\}_{i=1,\ldots n} \in \mathcal{H}$ which all have the same accuracy on D. Then by merging the output of the classifiers $\{H_i\}_{i=1,\ldots n,}$ the obtained ensemble classifier reduces the risk of choosing a single classifier that may have poor performance on new data. Second reason (*computational*): many learning algorithms have a stochastic component and perform some sort of random initialization search in $\mathcal{H}$. These algorithms can remain stuck in a local optima. An ensemble constructed by running the learning algorithm using different initializations may provide a better approximation of F. The third reason (*representational*): in many real cases $F \notin \mathcal{H}$. Hence by combining the different hypothesis $\{H_i\}_{i=1\ldots n} \in \mathcal{H}$, it may be possible to "expand" the space $\mathcal{H}$ and find a solution which is closer to F (Dietterich, 2000). In this study, random forest (RF), decision trees (DTs) and support vector machine (SVM) are used as classifiers. They are discussed in the next section.

2.4.4.1 Random Forest

A Random Forest (RF) is a combination of tree predictors wherein each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. Generally, RFs are less intensive in computation and proved to be comparable to boosting. Additionally, RFs can also be considered an improved version of bagging. An RF is a classifier consisting of a collection of tree-structured classifiers $\{h(\mathrm{x},\Theta_k),\ k=1,\ldots\}$ where the $\{\Theta_k\}$ are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input x (Breiman,2001). An RF algorithm processes large amounts of data and uses a huge number of trees in ensemble classifiers. This, combined with the fact that the random selection of variables for a split seeks to minimize the correlation between trees in the set, gives error rates that have been compared to those of boosting, while being much spacious in computation (Freund and Schapire, 1996).

The analysis of RF proves that its computational time is $cT\sqrt{MN}\log(N)$ where c is a constant, T is the number of trees in the ensemble, M is the number of variables and N is the number of samples in the data set. Although RFs are not computationally intensive, they require a fair amount of memory as they store an $N \times T$ matrix in memory (Breiman, 2003). Generally, RFs are based on the strength of the individual tree classifiers and the correlation between them. For classification, each tree in the RFs is illustrated by a unit vote for the most popular class at entry x. The output of the classifier is formed by a majority vote of trees. Practically, the RF algorithm has several advantages such as training speed, robustness and classification accuracy. It also provides useful internal estimates of error, the strength of the correlation variable and the importance of bagging or strengthening (Breiman, 2001).

Typically, an RF algorithm runs without setting parameters and provides a numerical estimate of the size of the feature. It is an algorithm wherein the classification is carried out on the vote of multiple weak, unbiased classifiers and trees; these trees are independently constructed on different bagging samples from the training set (Kursa and Rudnicki, 2010). Bagging shows the learning algorithm with a training set provided with a sample of S randomly drawn with replacement, where S is the size of the original training set. With bagging, many of the original instances may be repeated in the resulting training set while others may be left out (Breiman, 1996).

Table 10 Hypothetical run of bagging (Efron and Tibshirani, 1993)

	(Resampled) **Training set**							
Training-set-1 :	2,	7,	8,	3,	7,	6,	3,	1
Training-set-2:	7,	8,	5,	6,	4,	2,	7,	1
Training-set-3:	3,	6,	2,	7,	5,	6,	2,	2
Training-set-4:	4,	5,	1,	4,	6,	4,	3,	8

Table 10 shows a hypothetical run of bagging from a sample of a single classifier on an imaginary set of data where the original training set is 1,2,3,4,5,6,7,8. Parallelism is used in the bagging. In this scenario, different data models and outputs of all the predictors are built by each algorithm to produce the final output of the ensemble. In order to build different models, either each algorithm of the ensemble, or the data fed to each algorithm, or both, can be different. Since all algorithms perform in parallel, each set of the algorithm may be different, each of them can be run on a different processor to speed up the calculation and build different models. The parallel execution is an important advantage compared to the boosting method. The only additional time required is used for the decision function that combines the outputs of all the algorithms. Thus, ensemble does not increase the processing time significantly compared to an individual or single algorithm (Balon-Perin and Gambäck, 2013).

Breiman (1996) illustrates tree bagging as follows:
Considering a training set $(\boldsymbol{x_i}, \boldsymbol{\omega_i})$ for $\boldsymbol{i}$ = 1,2,..., N, where $\boldsymbol{x_i}$ is a $\boldsymbol{p}$ - dimensional vector and $\boldsymbol{\omega_i}$ indicates the target label of $\boldsymbol{x_i}$. Tree bagging repeatedly selects a bootstrap sample of the training set, which consists of $\boldsymbol{N}$ samples, and then fits $\boldsymbol{B}$ number of trees to these samples. The algorithm for tree bagging can be explained as follows (Breiman, 1996):

Input(s): Training set (x_i, ω_i) for i = 1,2, ..., N; Number of trees B
Output: Tree classifier $T_1, T_2, \ldots, T_B$
1. **for** b = 1 to B do
2. Build a dataset S_b, by sampling N items randomly with replacement from the original data pool (x_i, ω_i) for i = 1,2,...,N.
3. Train decision tree T_b using S_b and save it.
4. **end for**

Particularly, the only difference between bagging and the RF algorithm is that the RF algorithm considers a random subset of the features to be selected for splitting the node in the tree building process. Whereas bagging searches for the best way of splitting the feature set, considering the whole feature set. Correlation between trees in an ordinary bootstrap sample is reduced through the RF algorithm. If one or more feature sets have strong predictive power for the target variable using bagging, they will be selected in many of the DTs causing high correlation and reducing the power of the ensemble. Generally, the steps of a typical RF can be illustrated as follows (Liaw and Wiener, 2002):

1. Draw n_{tree} bootstrap samples from the original data.
2. For each of the bootstrap samples, grow an unpruned classification tree, with the following modification: at each node, rather than choosing the best split among all predictors, randomly sample m_{try} of the predictors and choose the best split from among those variables. (Bagging can be considered as the special case of RF obtained when m_{try} = p, the number of predictors.)
3. Predict new data by aggregating the predictions of the n_{tree} trees such as majority votes.

Moreover, the RF pseudo code can be illustrated as follows (Breiman, 2001):

Input(s): Training set (x_i, ω_i) for i = 1,2, …, N; Number of trees B ; number of features F
Output: Tree classifier $T_1, T_2, \ldots, T_B$
1. **for** b = 1 to B do
2. Build a dataset S_b, by sampling N items randomly with replacement from the original data pool (x_i, ω_i) for i = 1,2,…,N.
3. Train decision tree T_B without pruning using S_b . In each node splitting process, randomly select F features without replacement from the whole feature set and search the best splitting threshold inside the selected feature subset.
4. **end for**

The new test point classification consists of voting by the majority of decision trees B, similar to bagging trees. The number of trees B to build must be specified in advance, both for bagging trees and RFs. The specification of the parameters can be performed using an out-of-bag (OOB) error. The definition of an OOB error is the mean prediction error on each training sample x_i, using only the trees that did not have x_i in the bootstrap sample (Breiman, 2001).

2.4.4.2 Decision Tree

Decision tree (DT) is a widely-used method for ML (Breiman, 1996). A DT consists of nodes, leaves, and edges. The attributes of the samples are assigned to each node; the value of each branch is corresponding to the attributes (Mitchell, 1997). Each node in the tree specifies some test of problem attributes, represented by the *p+1* element vector (f_1, f_2,f_3, ...,f_n, class), where *n* shows the number of problem attributes. Each branch descending from that node matches one of the possible outcomes of that test. The advantage of DTs is that they can be generated and explained easily.

Figure 15 illustrates an example of a DT where every branch leading from the root (R) of the tree to the leaves (A,B,C) can be represented as

IF {Conditions} THEN {Class} rule

Where the *IF* clause includes the conjunction of the conditions derived from the nodes (f_1, f_2,f_n) and the outcome is the class of the leaf. The *IF* part of the rule is also known as (Papamartzivanos et al., 2018):

- antecedent : <*feature, operator, value*> and;
- the prediction part of the rule as consequent : *IF* { f_1 is V_1 ∧...∧ f_n is V_n} *THEN* Class = *C*.

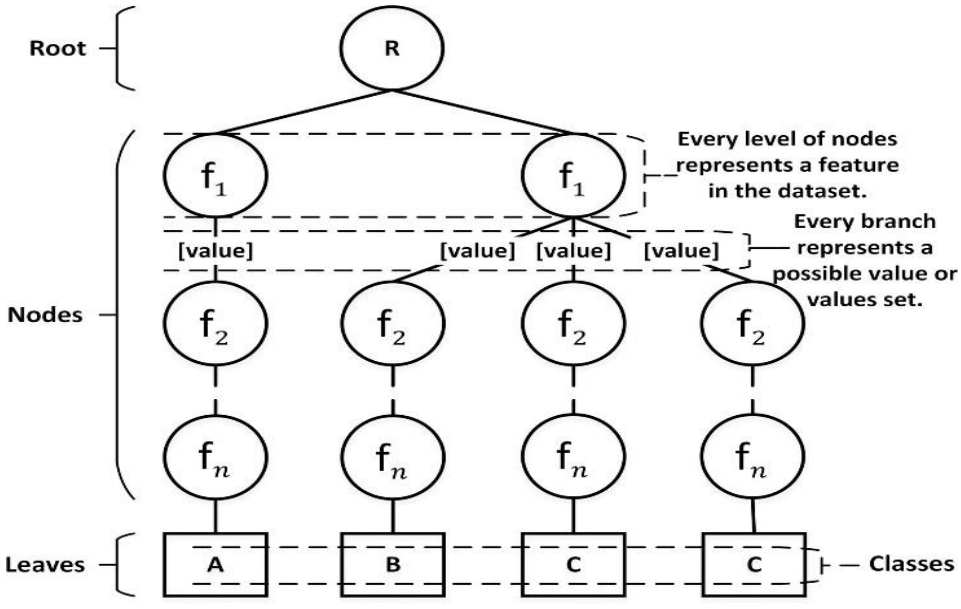


Figure 15 Structure and properties of a DT (Papamartzivanos et al., 2018).

A technique widely used to describe the order of attributes in the tree is derived from the information theory of calculating the information gain of each attribute and classifying the attributes or variables according to certain values. The root is illustrated by the variable with the highest information gain, the second variable having the highest information gain is the node, and so on. For a binary tree, the information gain can be calculated using the following equation (Balon-Perin, 2012):

$$gain\ (A) = I\ (p, n) - E\ (A) \tag{2.2}$$

Where

- E (A) is the expected information required for the tree with A as root.

$$E\ (A) = \sum_{i=1}^{v} \frac{pi+ni}{p+n} I\ (p_i, n_i) \tag{2.3}$$

Where

p is the number of positive examples, n is the number of negative examples, p_i is the number of positive examples for the i_{th} branch, n_i is the number of negative examples for the i_{th} branch and v is the number of branches from the root *A.*

- I(p,n) is the information required for classification:

$$I\ (p, n) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n} \tag{2.4}$$

Where

p is the number of positive examples and n is the number of negative examples.

In addition, understanding the criteria for splitting nodes before analysing the algorithm is of paramount importance. By optimizing the cost function for each node, the subset and corresponding threshold of the feature and variables are defined. During the testing phase, variables are classified as either the right child node or the left child node, based on the value of the feature. A value greater than the threshold belongs to the right child node and a value less than the threshold to the left child node (Breiman, 1996).

Gini impurity, $I_G(f)$ and Entropy, $I_E(f)$ are traditionally employed to select the feasible splitting feature and corresponding threshold. The definitions are as follows (Witten *et al.*, 2011):

$$I_G(f) = -\sum_{i=1}^{m} f_i(1 - f_i) = 1 - \sum_{i=1}^{m} f_i^2, \tag{2.5}$$

Figure 16 illustrates the plots for binary classification. Gini impurity and entropy define measures of similarity on a target feature. Gini impurity and entropy approach 0 where f approaches 0 or 1. This indicates that illustrations are equivalent and are from the same class. The feature and threshold that will cause the largest fall of these values between the parent and child nodes are selected to split this node. The aim is to maximise the difference and is called information gain (Breiman,1996).

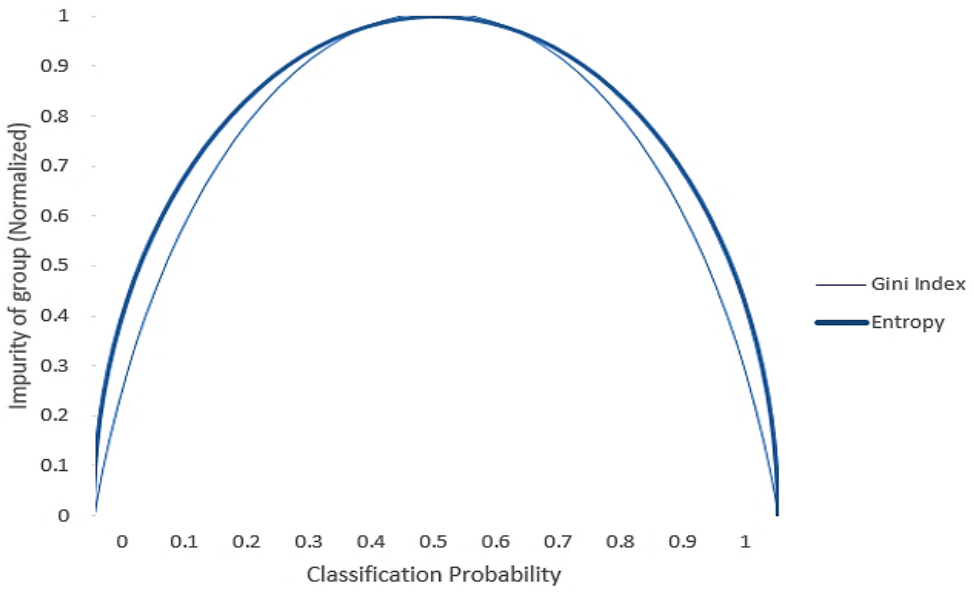


Figure 16 Gini impurity and entropy curves (Witten et al., 2011)

Considering a set of variables with m classes, $i \in \{1,2,\dots,m\}$ and f_i the fraction of items labelled with class i in the set, and

$$I_E(f) = -\sum_{i=1}^{m} f_i \log_2 f_i \quad , and \tag{2.6}$$

$$\sum_{i=1}^{m} f_i = 1\,. \tag{2.7}$$

Eq. (2.7) shows the percentage of each class present in the child node that results from a split in the tree (Witten et al., 2011).

In addition, the entropy method is used to calculate the gain for ID3 and C4.5. The classification and Regression Tree (CART) method of Breiman (1996) is another widely used method of analysis that uses Gini impurity to measure homogeneity. Minimal cost-complexity pruning avoids over-adjustments and is part of many methods, including the CART method (Mansour, 1997). The purpose of this step is to build a right-sized tree by approximating the true misclassification cost. The CART method firstly builds a fully-grown tree and then cuts the pair of leaves. The value of cost-complexity and the cost of classification errors are computed using cross-validation for each subtree. The final optimal tree is then identified based on the final values produced by the CART algorithm (Breiman, 1996).

In particular, it has been proven that DT gives a higher score to variables with many values, this does not mean that these variables are relatively important. Many algorithms have been implemented to build DTs; Smaller DTs are often more accurate because they have a strong generalization power than complex trees (Balon-Perin, 2012). The DT is constructed during the learning stage and then used to predict the classes of new instances through various strategies. Practically, one method used to find high precision hypotheses is based on the pruning of rules from the tree built during the learning stage (Bouzida and Cuppens, 2006). The pruned rules have many advantages in NIDS. Rules such as C4.5 and C50 have several attributes for the NIDS because they generate a perfect generalization precision (Bouzida and Cuppens, 2006). The rule post pruning can be illustrated as follows (Bouzida and Cuppens, 2006):

IF *(protocol type = icmp)* $\wedge$ *(count> 87)*

THEN *class = smurf*

Moreover, DT presents several advantages over other ML techniques as detailed below (Quinlan, 1987):

1) It is simple to understand and interpret.
2) Data preparation is minimal. Other methods often require normalization of data, replication of variables and deletion of empty values.
3) It uses a white box model. The classification model is clear and explicit; the variables are associated with the result of the tree structure.
4) It is a robust and non-parametric classifier. The decision tree method is not a simple classifier, it also serves as a basis for many ensemble classifiers.

2.4.4.3 Support Vector Machine

The support vector machine (SVM) is a binary classifier proposed by Vapnik and his group at AT&T Bell Laboratories (Cortes and Vapnik, 1995). In this work, SVM is proposed to enhance the robustness and reliability of the NIDS. Mostly, when implementing the NIDS-based ensemble, each classifier must be different from each other. This is based on the uniqueness that presents each algorithm to produce good results. There are usually methods to combine the individualities presented by each classifier; among them, the focus is on fusion methods such as majority voting and weighted majority voting (they are discussed in the next chapter).

In general, SVM creates a hyperplane or multiple hyperplanes in a high-dimensional space, and the best hyperplane in them is the one that optimally divides data into different classes with the largest separation between the classes. In order to estimate the margins, a non-linear classifier uses various kernel functions. The main objective of these kernel functions is to maximize the margins between hyper-planes (Ahmad et al., 2018). The Gaussian RBF, linear kernel and polynomial kernel are among the most kernel functions used (Achirul et al., 2018).

The following equations represent the mathematical formulation of linearly separable setting:

Considering a training dataset P, a set of n points of the form

$$P = \{(x_i, \omega_i) | x_i \in \mathbb{R}^p, \omega_i \in \{-1, 1\}\}, i = 1, \dots, n \ , \tag{2.8}$$

Where ω_i is either 1 or -1, indicating the label of observation x_i. Each x_i is a p-dimensional feature vector. The goal is to find a hyperplane that maximises the margin between the points, having ω_i = 1 and ω_i = −1, where any hyperplane can be written as the set of points x satisfying:

$$w \cdot x - b = 0 \ , \tag{2.9}$$

b shows the distance of one from the hyperplane to the closest points in each class.

The $\cdot$ indicates the dot product and w the normal vector to the hyperplane. The expression $\frac{b}{\|w\|}$ determines the space between the hyperplane and the origin.

Additionally, SVM can be considered as a more elaborated version of logistic regression and the generator of binary values. It is therefore possible to use SVM as a multiclass classifier; a hypothesis is then formulated to generate 0 or 1. Figure 17 illustrates the logistic function:

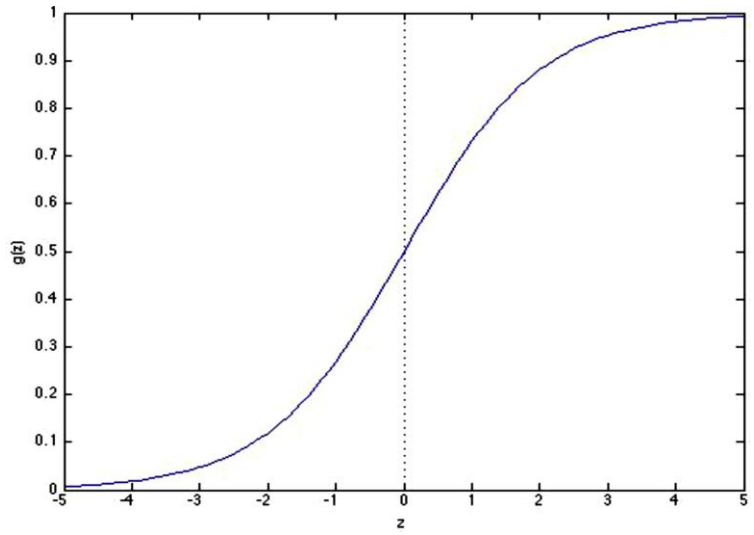


Figure 17 Logistic function (Balon-Perin, 2012)

$$g\,(z) = \frac{1}{1+\exp(-z)} \tag{2.10}$$

In ML based-ensemble technique, the logistic function is slightly modified as follows (Balon-Perin, 2012):

$$h_\theta\,(x) = \frac{1}{1+\exp(-\theta^T x)} \tag{2.11}$$

Where

- x is a matrix of dimension m×n where m is the number of examples in the dataset and n is the number of variables in the dataset.
- θ is a vector of parameters of dimension n×1 where n is the number of variables in the dataset. The goal of the training phase of ensemble learning is to determine the optimal values for this vector of parameters.
- $h_\theta(x)$ is the hypothesis and represent the value that the classifier predicts for a particular example x^i (Balon-Perin, 2012).

Generally, two support vectors are selected to separate the dataset, without any intermediate point. The space between the support vectors must be maximized in order to separate the data in a linear fashion. The margin is defined as the region bound by the two support vectors. The distance between two or more support vectors can be represented in various ways (Burges, 1998). Figure 18 illustrates the interval between two support vectors.

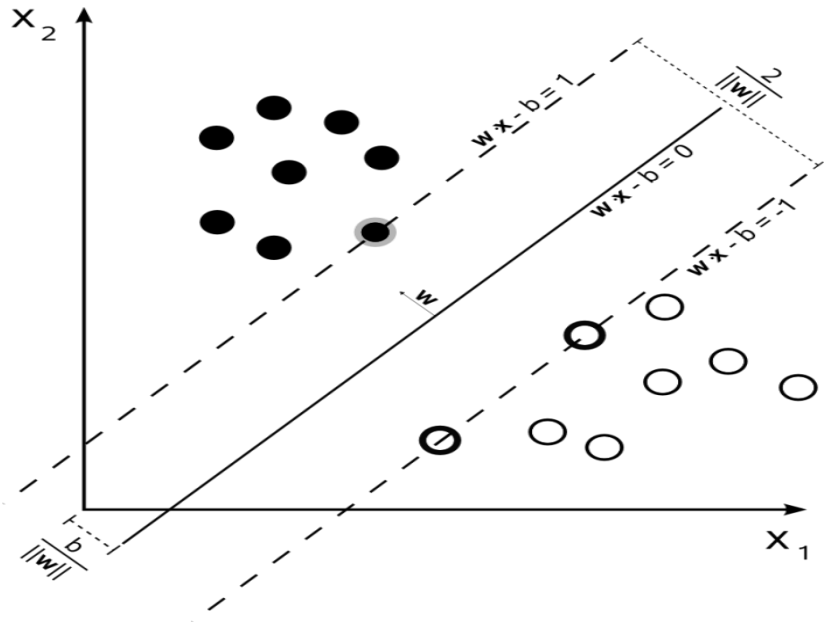


Figure 18 SVM classification boundary (Burges, 1998)

The space between the two support vectors can be represented as $\frac{2}{\|\boldsymbol{w}\|}$, hence $\|\boldsymbol{w}\|$ must be minimised. Initially, the minimization of structural risk determines the SVM, which reduces the error of generalization. The number of free spaces used in the SVM indicates the margin separating the data points but not necessarily the number of input features. SVM does not require a reduction in the number of features to avoid over-adjustments or overfitting; it provides a common mechanism which adjust the hyperplane space to the data by using a kernel function. Then, during the training stage, several functions (SVM linear, polynomial or sigmoid curves) are provided by setting support vectors along the space or surface of these functions. This helps to classify an extensive space of problems (Mukkamala et al., 2002). From figure 18, the filled (class label = "1") and unfilled (class label = "-1") circle points indicate training points belonging to various classes. The two dashed lines indicate the boundary of maximum margin, while the solid line shows the classification boundary (Burges, 1998).

The following constraints prevent data points from falling into the margin (Burges, 1998):

$$w \cdot x_i - b \geq 1 \text{ for } x_i \text{ having label "1" and} \tag{2.12}$$

$$w \cdot x_i - b \leq -1 \text{ for } x_i \text{ having label "-1".} \tag{2.13}$$

These constraints (Eq. 2.12 and 2.13) could be further minimized to:

$$\omega_i(w \cdot x_i - b) \geq 1 \text{ for all } 1 \leq i \leq n, \tag{2.14}$$

Where n is the set of points.

To summarise, the Eq. (2.14) becomes:

$$\arg \min_{w,b} \max_{\alpha \geq 0} \left\{ \frac{1}{2} \|w\|^2 - \sum_{i=0}^{n} \alpha_i [\omega_i (w.x_i - b) - 1] \right\}. \tag{2.15}$$

Where $\alpha_i, for\ all\ i = 1, \dots, n$ are positive Lagrange multipliers. Hence minimising $\|w\|$ equals reducing $\frac{1}{2}\|w\|^2$.).

Typically, the above equation can be interpreted and solved by using the Karush-Kuhn-Tucker condition and the standard quadratic programming methods. Slack variable, ξ_i, is illustrated to the objective function for linear inseparable problems; it becomes (Fletcher, 2013):

$$\arg \min_{w,\xi,b} \max_{\alpha \geq 0, \beta \geq 0} \left\{ \frac{1}{2} \|w\|^2 + C \sum_{i=0}^{n} \xi_i - \sum_{i=0}^{n} \alpha_i [\omega_i (w.x_i - b) - 1 + \xi_i] - \sum_{i=0}^{n} \beta_i \xi_i \right\}, \tag{2.16}$$

Where C is a vector of constraint functions and $\beta_i \in \mathbb{R}^p$. The tolerance of the misclassification of the model is measured by the slack variable ξ_i. If a problem is linearly inseparable, the Kernel trick can be used in order to solve it. Then the input data is depicted to a higher dimension or infinite dimension feature space (Aizerman, 1964). Generally, SVMs are considered the reference in many comparative classification studies because of their robustness and performance.

2.5 Summary

The purpose of this chapter was to review the literature on the NIDS, to evaluate its historical development, and bring a definition that serves as the theoretical basis of this work. This literature review has provided a detailed discussion of the key concepts used in the field of NIDS. In this chapter, ML techniques, various categories of attack and ensemble-based classifiers (RFs, DTs & SVM) were also explained. In the next chapter (chapter 3), the method applied in this research is explained.

Chapter 3-Methodology

A solid foundation in genetics is increasingly important for everyone…

-Anne Wojcicki-

This chapter provides general methodology that is be applied to solve the NIDP and to build a framework by describing specific procedures of ensemble learning.

3.1 Research approach

The approach aims at developing a NIDS based-ensemble technique that would be more accurate and low in false alarms (for probe, U2R, L2R, normal and DoS classes). Prior to data analysis, the features extracted from the input packets are established, relevant attributes are defined and a classifier is used to compute the variables. Acceptance of variables for each class as specified by the control variables is established by the data processor. In order to create an ensemble of precise and varied classifiers, the methods are based on training tree classifiers using the same training sets, but with several subsets of features. These classifiers are executed several times, each time with a different subset of the training dataset.

The research approach to solve the NIDP is illustrated as follows:

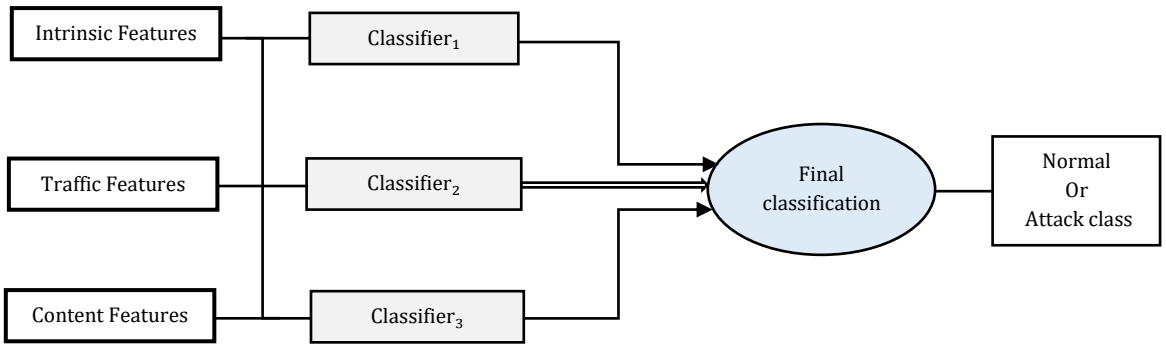


Figure 19 Proposed ensemble approach to solve the NIDP

3.2 Proposed methodology

Figure 20 illustrates the NIDP methodology proposed in this work:

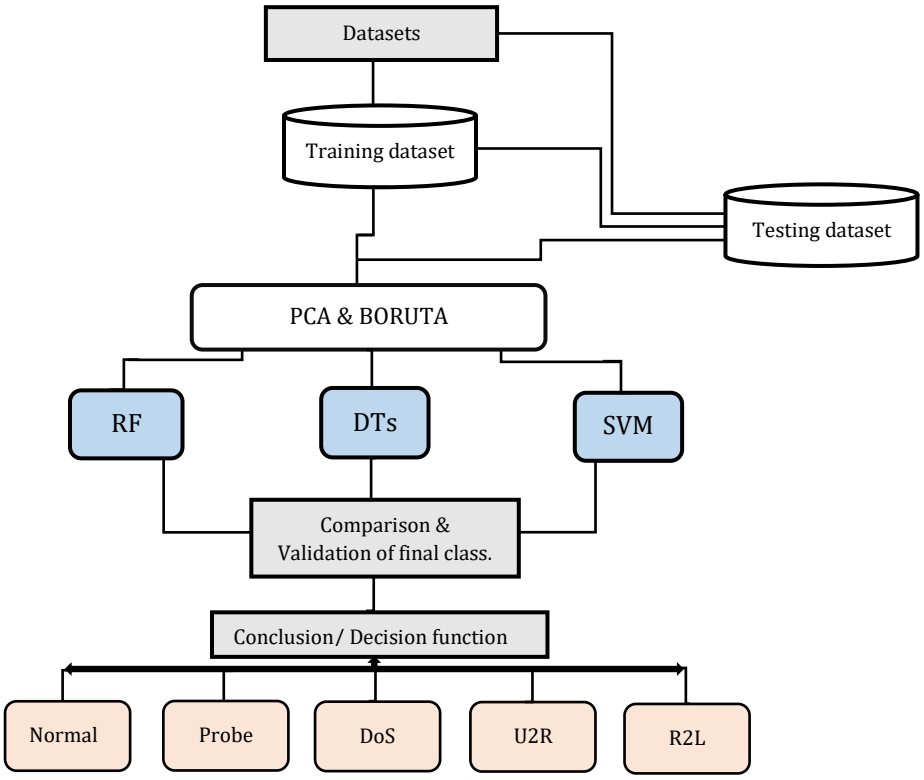


Figure 20 Proposed research methodology

The datasets will be used from various ML data libraries. The KDD99 and NSL KDD datasets derived from the University of California, Irvine ML repository and the University of New Brunswick (Staudemeyer and Omlin, 2014; Fan et al., 2000). They have been extensively used in the field of NIDS. These datasets provide a representative baseline for evaluation of the NIDSs (Creech and Hu 2013); and will assist to test and validate the system. Hence, to evaluate the three classifiers (RFs, DTs & SVM), these datasets will be applied to assess the system performances (prediction, accuracy rate, false positive and true positive) comparing to the works conducted by Balon-Perin (2012) and Zainal et al. (2009). The next section discusses the feature selection algorithms. Datasets are explained in chapter 4 (section 4.3).

3.3 Feature selection

Feature selection is an ML technique used for reducing dimensionality among practitioners; it chooses a small subset of the relevant attributes from the original ones based on certain evaluation criterion, which generally leads to better performance, lower computational cost, and better model interpretability (Tang et al., 2014). Feature selection techniques can be subdivided into supervised (filter models, wrapper models and embedded models), unsupervised and semi-supervised feature selection (Song et al., 2007). The filter model divides feature selection from classifier learning and reduces the bias of both learning and feature selection algorithms by relying on measures of the attributes such as consistency, dependency, information, and correlation (Robnik-Šikonja and Kononenko, 2003).

Supervised feature selection evaluates the relevance of the features guided by the tag information while unsupervised feature selection works with untagged data and semi-supervised feature selection uses labelled and unlabelled data to estimate the relevance of feature. The wrapper model uses the accuracy of a predefined learning technique to estimate the effectiveness of selected features. Generally, the embedded model achieves accuracy and efficiency comparable to that of the filter model (Robnik-Šikonja and Kononenko, 2003).

Table 11 Algorithms of feature selection for classification (Tang et al., 2014).

Feature Selection		
1. Flat features Algorithms	**2. Streamings features Algorithms**	**3. Structured features Algorithms**
1.1 Filter models	**2.1** Grafting Algorithm	**3.1** Group structure
1.2 Wrapper models	**2.2** Alpha-investing Algorithm	**3.2** Tree structure
1.3 Embedded models	**2.3** Online Streaming Feature Selection Algorithm	**3.3** Graph structure

Table 11 depicts the classification of feature selection algorithms for classification. Generally, feature selection for classification can be subdivided into three groups namely techniques or algorithms for flat features where features are assumed to be independent, streaming features algorithms where features are generated dynamically and whose size is unknown and algorithms for structured features where features are independent and totally overlook the feature structures (Tang et al., 2014).

Generally, feature selection technique consists of 4 main steps (Liu and Yu, 2005):

1. Subset generation: a candidate feature subset is selected based on a given search strategy.
2. Subset evaluation: evaluates the feature subset based on certain evaluation criterion such as accuracy, etc.
3. Stopping criterion: chooses the best fits of the evaluation criterion from all the candidates being evaluated and stop when criterion are met.
4. Result validation: validates the selected subset using a validation set.

The ultimate goal of feature selection algorithm is to enhance learning performance, to reduce computational complexity by building better generalizable models and reducing required storage of features. Let Y be the original set of features, with cardinality n. Let d represents the desired number of features in the selected subset X, $X \subseteq Y$. Let the feature selection criterion function for the set X be represented by $J(X)$. Without any loss of generality and considering a higher value of J to show a better feature subset. Since $J(.)$ is being maximized, one feasible criterion function is $(1 - p_e)$, where p_e denotes the probability of error. Using the error probability as a criterion function makes the selection of characteristics dependent on the specific classifier used and the size of the training and test data sets (Jain and Zongker, 1997).

Formally, by finding a subset $X \subseteq Y$ such that $|X| = d$, the feature selection can be formulated as follows (Jain and Zongker, 1997):

$$J(X) = \max_{Z \subseteq Y, |Z|=d} J(Z). \tag{3.1}$$

A comprehensive approach to the above formula would require consideration of all $\binom{n}{d}$ feasible d-subsets of the feature set Y. The number of possibilities increases exponentially, making an exhaustive search impossible even for moderate values of n. Cover and Van (1977) showed that no non-exhaustive sequential feature selection procedure can be guaranteed to generate the optimal subset. They also showed that any order of the probabilities of error of the 2^n subsets of variables is possible (Cover and Van, 1977).

Moreover, the extraction of appropriate variables or features illustrating the NIDS is based on a thorough knowledge of the characteristics that distinguish intrusions from normal connections. The main features are (Lee and Stolfo, 2000):

1. Intrinsic features (basic features): include the duration, type, protocol, flag, etc. of the connection.
2. Traffic features: contain statistics related to past connections similar to the current one (number of connections with the same destination host or connections related to the same service in a given time window or within a predefined number of past connections).
3. Content features : contain details about the data content of packets that could be relevant to discover an intrusion

Figure 21 illustrates the features of the NIDS (Lee and Stolfo, 2000)**:**

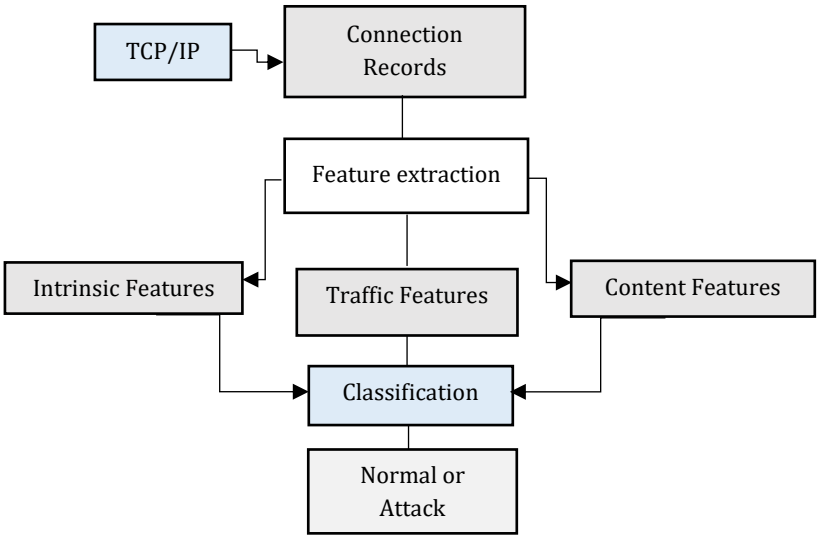


Figure 21 Features of NIDS (Lee and Stolfo, 2000)

The classification is based on classifying each connection record into intrusions (probe, U2R, L2R, normal and DoS classes) or normal instances. Each connection record or host session contains a considerable set of features and appropriate variables describing the characteristics of the three features (basic, traffic and content features); duration, type, number of packets and bytes, etc. The next section introduces the PCA and Boruta algorithms as main feature selection algorithms applied in this work.

3.3.1 Principal Components Analysis

Principal components analysis (PCA) was invented in 1901 by Karl Pearson (1901). PCA is well known as a technique for data compression and feature selection (Oja, 1992). PCA is widely used by NIDSs to extract unseen features from intrusion patterns. The PCA aims to identify the subset of intrusions by training various variables and to decrease the dimensionality of the dataset while keeping much of the initial variability in the data (Bouzida et al., 2004). PCA can be formulated as follows (Liu and Yang, 2007):

Assuming that {x(t)} t= 1, 2…l, are n-dimensional stochastic input data with zero mean and the covariance matrix of x(t) is defined by

$$R=\frac{1}{l-1}\sum_{t=1}^{l}[x(t)x(t)^{T}] \quad (3.2)$$

Then PCA can transform x(t) into y(t) linearly by

$$y(t)=Q^{T}\,x(t), \quad (3.3)$$

Where Q is an n×n orthogonal matrix whose i_{th} column is the i_{th} eigenvector of the sample covariance matrix R. PCA firstly solves the following eigenvalue problem:

$$\lambda_i\, q_i = R q_i \quad (3.4)$$

Where λ_i is one of the eigenvalues of R and q_i is the corresponding eigenvector. From (3.3), it may be written as follows:

$$y_i(t) = Q^{T}\,x(t), \qquad i = 1,\ldots,n. \quad (3.5)$$

The new component y_i refers to the i_{th} principal component. While sorting the eigenvalues λ_i in descending order, m is obtained from the eigenvectors corresponding to those m largest eigenvalues (usually $m<n$) for feature extraction. The variance of the input data projections on the main direction is well known to be greater than any other direction. Giving a new input x(t), it is projected onto the principal space which is composed of the first m principal directions, B = {b_i|b_i = q_i, i= 1,…,m}(Liu and Yang, 2007). The eigenvectors represent a set of features that together characterize the variation between records connections (Bouzida et al., 2004).

3.3.2 Boruta Algorithm

The Boruta algorithm is a wrapper model built around the RF algorithm implemented in the R package randomForest (Liaw and Wiener, 2002). Boruta is a continuation of the idea illustrated by Stoppiglia et al. (2003) explaining relevance by comparing the relevance of actual variables to the random probes (Stoppiglia et al., 2003). Before being considered as a wrapper model, Boruta was introduced in the context of filtering. Boruta uses techniques such as the standard score (Z-score) to measure the average precision and the loss of trees in the forest. For each attribute of datasets, Boruta generates a correlative "shadow" attribute, whose values are acquired by mixing the values of the original attributes between the objects. Then, a classification is performed using all the attributes and calculating the importance of all features (Kursa and Rudnicki, 2010).

Boruta follows the steps below (Kursa and Rudnicki, 2010):

1. Extending the information system by adding copies of all variables.
2. Shuffling the added attributes to remove their correlations with the response.
3. Executing the RF classifier on the extended information system and gathering the Z scores computed.
4. Finding the maximum Z score among shadow attributes (MZSA), and then assigning a hit to every attribute that scored better than MZSA.
5. For each attribute with undetermined importance perform a two-sided test of equality with the MZSA.
6. Consider the attributes which have importance significantly lower than MZSA as "unimportant" and permanently remove them from the information system.
7. Consider the attributes which have importance significantly higher than MZSA as "important".
8. Remove all shadow attributes.
9. Repeat the procedure until the importance is assigned for all the attributes, or the algorithm has reached the previously set limit of the RF runs.

Sample code of Boruta algorithm in Rstudio:

```
boruta.train <- Boruta(result~.-service, data = data,doTrace = 2,maxRuns=25)
print (boruta.train)
```

3.4 Ensemble combination strategies

There exist many methods (manipulating the training set, input features, class labels and the learning algorithm) for constructing ensemble classifiers. This section covers general ensemble methods and combination strategies. Initially, the procedure of building an ensemble can be illustrated as follows (Rokach, 2010):

Let S denote the original training data, n denote the number of base classifiers, and T be the test data.

For i = 1 to n **do**

Create training set S_i from S.

Build a base classifier C_i from S.

end for

for each test record $x \in T$ **do**

$C^*(x) = \text{Vote}\ (C_1(x),\ C_2(x)\ldots,\ C_n(x))$

end for

Where C^* is an ensemble of classifiers and C_i is the i^{th} classifier in ensemble C^*. The vote method is the weighting of outputs from each classifier (C_1,C_2…,C_n). Thus, in ensemble method it is necessary to find out the eligible classes for which a classifier is most likely to vote (Saha and Ekbal, 2013). A fusion method or combination strategies yield the best decisions and eligible classes. In combination strategies, all classifiers are trained on the entire feature space, then combined to obtain a composite classifier or base inducer with lower variance. The decisions produced by the base inducer or classifiers are combined into a single decision using various strategies (Polikar, 2012).

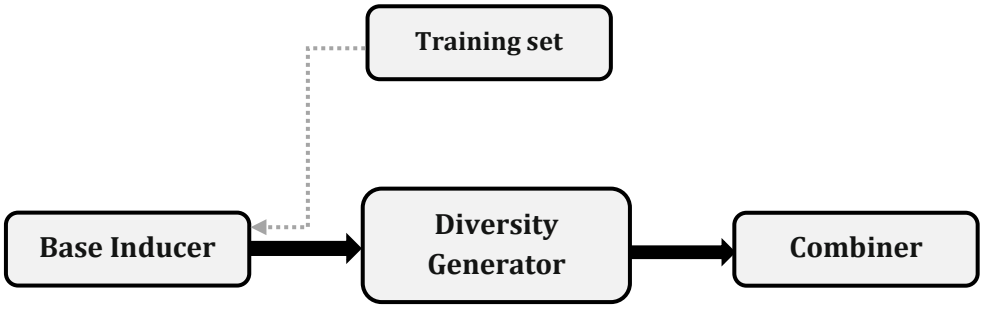


Figure 22 Buildings blocks of the ensemble method (Rokach, 2010)

Figure 22 shows the components of an ensemble classification method (Rokach, 2010). The next section covers the two ensemble combination strategies (combining class labels and combining continuous outputs).

3.4.1 Combining class labels

Assuming that only class labels are available in the classifier outputs and defining the decision of the t^{th} classifier as $d_{t,c} \in \{0,1\}$, t =1,..., T and c = 1 ,..., C , where T is the number of classifiers and C is the number of classes. If t^{th}classifier (or hypothesis) h_t chooses ω_c , then $d_{t,c}$ = 1, and 0, otherwise. The most common combination rules for class labels are: majority voting and weighted majority (Polikar, 2012).

3.4.1.1 Majority voting

Majority voting is based on (Polikar, 2012):

- Unanimous voting: all classifiers agree on the ensemble decision of each class.
- Simple majority: predicted by at least one more than half the number of classifiers.
- Plurality voting: classifier that receives the highest number of votes, whether or not the sum of those votes 1 ensemble learning 7 exceeds 50%.

Typically, majority voting refers to plurality voting, which can be mathematically defined as follows: chooses w_{c*} , if

$$\sum_{t=1}^{T} d_{t,\,c^*} = max_c \sum_{t=1}^{T} d_{t,\,c} \tag{3.6}$$

Where T is an odd number of classifiers. Then, the ensemble gives the correct decision if at least $\lfloor T/2 \rfloor$ + 1 of these classifiers choose the correct label. The binomial distribution constitutes the accuracy of the ensemble; the probability of having $k \geq [T/2] + 1$ out of T classifiers returning the correct class. Since each classifier has a success rate of p, the probability of ensemble success is then

$$p_{ens} = \sum_{k=\frac{T}{2}+1}^{T} \binom{T}{k} p^k (1-p)^{T-k} \tag{3.7}$$

Note that p_{ens} approaches 1 as $T \to \infty$ 1, if $p > 0.5$; and it approaches 0 if $p < 0.5$ (Polikar, 2012).

3.4.1.2 Weighted majority voting

Weighted majority voting corresponds to the weighted decisions of classifiers which are more likely to be correct than others and more heavily to enhance the overall performance compared to plurality voting. Assuming a mechanism to predict the approximate generalization performance of each classifier. A weight w_t can be assigned to classifier h_t in proportion of its estimated generalization performance (Polikar R. 2012).The ensemble, combined according to weighted majority voting then chooses class c^*, if

$$\sum_{t=1}^{T} w_t d_{t,\,c^*} = max_c \sum_{t=1}^{T} w_t d_{t,\,c} \tag{3.8}$$

3.4.2 Combining continuous outputs

If a classifier gives a continuous output for each class (such as radial base function networks, relevance vector machines, etc.), this output, after appropriate normalization, is illustrated in the equation below (Duda and Stork, 2001) and can be interpreted as the degree of support given to this class, and under certain conditions, can also be explained as an estimate of the posterior probability for this class. Representing the actual classifier output corresponding to class ω_c for instance x as $g_c(\mathrm{x})$, and the normalized values as $\bar{\mathrm{g}}_c(\mathrm{x})$, later probabilities approximated $P(\omega_c|x)$ can be obtained as

$$P(\omega_c|x) \approx \bar{\mathrm{g}}_c(\mathrm{x}) = \frac{e^{g_c(\mathrm{x})}}{\sum_{i=1}^{C} e^{g_i(\mathrm{x})}} \Longrightarrow \sum_{i=1}^{C} \bar{\mathrm{g}}_i(\mathrm{x}) = 1 \tag{3.9}$$

Kuncheva's decision profile matrix $DP(\mathrm{x})$ is illustrated with a view to consolidating different combination rules, which elements $d_{t,c} \in [0, 1]$ show the support given by the t^{th} classifier to ω_c(Kuncheva, 2001). Specifically, as illustrated in figure 23:

- The rows of $DP(\mathrm{x})$ indicate the support given by individual classifiers to each of the classes.
- The columns indicate the support received by a particular class c from all classifiers.

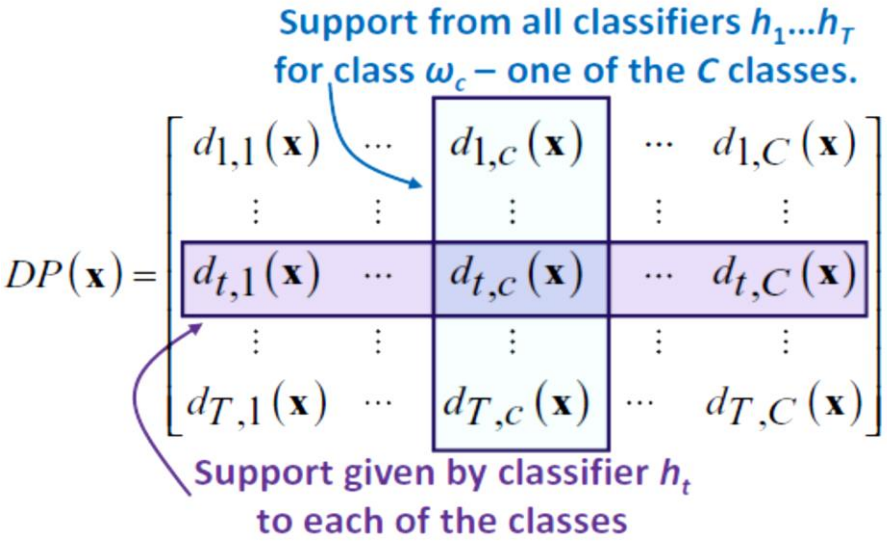


Figure 23 Decision profile for a given instance x (Kuncheva , 2001)

3.4.2.1 Algebraic combiners

In algebraic combiners, the total support for each class is obtained as a simple algebraic function of the supports received by individual classifiers. Following the notation used in the figure 15 (Decision profile for a given instance x (Kuncheva, 2001), the total support received by class $\boldsymbol{\omega_c}$ is the $\boldsymbol{c^{th}}$ column of the decision profile *DP*(*x*), as

$$\mu_{c,(x)} = F\,[d_{1,c\,(x)},\ldots,\, d_{T,C(x)}] \tag{3.10}$$

Where *F* […] is one of the following combination functions.

3.4.2.2 Mean rule

In mean rule, the support for class ω_c is the average of all classifiers c^{th}outputs; hence the function F [.] is the averaging function.

$$\mu_{c,(x)} = \frac{1}{T}\sum_{t=1}^{T} d_{t,c(x)} \tag{3.11}$$

The mean rule results in the identical final classification as the sum rule, which only differs from the mean rule by the 1/T normalization factor. In either case, the final decision is the class ω_c for which the total support $\mu_{c,(x)}$ is the highest (Polikar, 2012).

3.4.2.3 Weighted average

The weighted average merges the mean and the weighted majority voting rules in which weights are applied to class labels but to the actual continuous outputs. The weights are generated under training phase. Each classifier h_t receives a weight, although it is also feasible to assign a weight to each class output of each classifier (Polikar, 2012). Then T weights, $w_1, \ldots, w_T$ are usually obtained as estimated generalization performances based on training data, with the total support for class ω_c as

$$\mu_{c,(x)} = \frac{1}{T}\sum_{t=1}^{T} w_t\, d_{t,c(x)} \tag{3.12}$$

Afterwards, there are T * C class and specific weights, which leads to a class-conscious combination of classifier outputs (Kuncheva, 2001). Total support for class ω_c is then

$$\mu_{c,(x)} = \frac{1}{T}\sum_{t=1}^{T} w_{t,c}\, d_{t,c(x)} \tag{3.13}$$

Where $w_{t,c}$ represents the weight of the t^{th} classifier for classifying class ω_c

3.5 Summary

The purpose of this chapter was to introduce the method applied in this work and to explain the effectiveness of ensemble approach which depends on the choice of the fusion function. The decision function was based on the individual performances of each classifier and true positive rates of the network connection (normal or attack). Ensemble combination or fusion strategies (combining class labels & combining continuous outputs) were also discussed. In the next chapter (chapter 4), system framework using RStudio and final model are explained.

Chapter 4-Method and Framework

We become what we behold. We shape our tools, and thereafter our tools shape us…

-Marshall McLuhan-

Before delving deeper into various experiments and assessments, the system overview and final method are presented in this section. This chapter aims at building and implementing the NIDS based-ensemble methods in view to achieve the objectives set in Chapter 1.

4.1 System overview

RStudio will be used for implementing the NIDS; it is a programming language for statistical calculation, supervised and unsupervised ML tools for data, images, etc. RStudio was founded by Joseph J.Allaire (Chollet and Allaire, 2018). It also includes collections of prepared data for use in networks and computations.

4.1.1 System requirements

A certain number of requirements must be met when building the NIDS based-ensemble. Before listing the requirements, it is important to illustrate four essential terms (Balon-Perin, 2012):

- False Positive (FP): shows the number of instances that are classified by the NIDS as abnormal when they are legitimate. FP seems to be a negligible problem, but in practice the NIDS with a high FP rate will be as useless as the one with a high FN rate.
- False Negative (FN): represents the number of instances that are classified by the NIDS as being legitimate when they are abnormal. FN shows an attack that was misclassified by the NIDS as being a legitimate action.
- True Positive (TP): shows the number of instances that are classified by the NIDS as being abnormal and which are actually abnormal.
- True Negative (TN): represents the number of instances that are classified by the NIDS as being legitimate and which are actually legitimate.

In addition, the NIDS must meet the following requirements (Debar et al., 1999):

- Accuracy: also known as robustness, this property ensures that the system does not classify legitimate instances as abnormal.
- Performance: the system should be able to classify the traffic without adding significant overload to the network.
- Completeness: this property indicates that the NIDS should be able to identify all intrusion attempts resulting in a FN rate equal to 0. The completeness seems difficult to achieve because the NIDS must be able to identify unknown attacks.
- Fault Tolerance: the NIDS must itself be resistant to attacks.
- Scalability: a NIDS must be able to handle network traffic in real-time without losing packets, because its bandwidth is greater than what the NIDS can handle.

Furthermore, a good and reliable NIDS will attempt to compromise one of the following aspects (Fernandez and Porres, 2008):

1. Authentication: the system assures the recipient that the data or information comes from the source it claims to be.
2. Confidentiality: the intruder has no access to confidential information.
3. Integrity: information cannot be modified or altered by the attacker.
4. Availability: the system is operable and cannot be interrupted by any intrusion.

Figure 24 illustrates the four main aspects of a reliable NIDS:

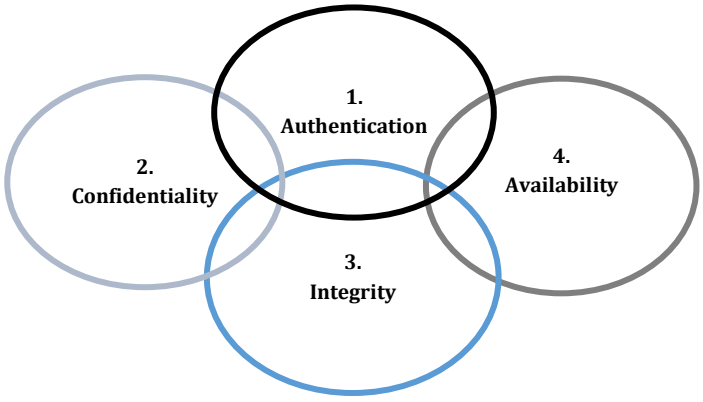


Figure 24 The four main compromised aspects (Fernandez and Porres, 2008)

Afterwards, the NIDS is monitored to identify and exclude possible attacks. NIDS is then composed of three main blocks (Fernandez and Porres, 2008):

1. Information source (providing useful information).
2. Analysis (engine looking for possible intrusions).
3. Response mechanism (manual or automatic).

Additionally, the NIDS's audit trail follows the steps below:

- Source of anomaly: determines the origin of the violation and the source of anomalies or intrusions.
- Damage verification or evaluation: analyses, investigates and checks for damage in the system and how it was done.
- System recovery: actions leading to the operable and normal state of the NIDS.

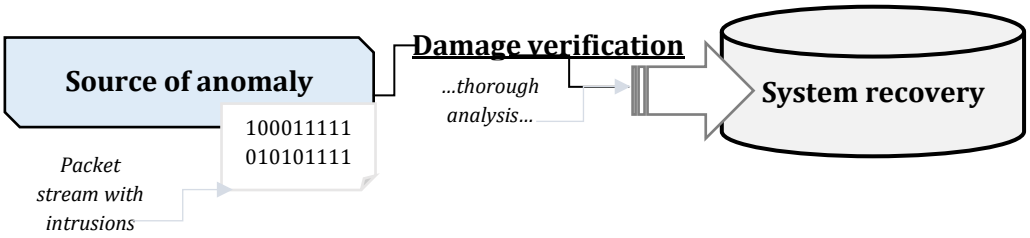


Figure 25 Audit trail of the NIDS

Figure 25 illustrates an audit trail of the NIDS. An audit trail is the process of examining system events to ensure that security policies are not violated (Fernandez and Porres, 2008).

4.1.2 System evaluation and validation

The above requirements imply that the NIDS must have both a FP rate and a FN rate equal to zero. Hence the system will be mainly evaluated and validated in two ways:

- The overall results from RF, DTs and SVM will be compared to a selected benchmark NIDSs as defined by Balon-Perin (2012) and Zainal et al. (2009).
- The system results in the NIDS library will be used to validate results obtained from the applied system. Thus the final classification will reveal which method can substantially improve ensemble classifiers and obtain the best performance in terms of accuracy.

The performance of the NIDS will be evaluated by calculating measurements from the values of the confusion matrix presented in table 12. The confusion matrix shows the distribution of correctly classified or misclassified instances by the classifier (Balon-Perin, 2012).

Table 12 Confusion matrix (Balon-Perin, 2012)

		Actual class	
		Negative class (Normal)	Positive class (Attack)
Predicted class	Negative class (Normal)	True Negative (TN)	False Negative (FN)
	Positive class (Attack)	False Positive (FP)	True Positive (TP)

The following measures are often used to assess the NIDS's performance (Balon-Perin, 2012):

- Accuracy: $Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$
- False positive Rate (FPR) or False Alarm Rate (FAR): $FPR = \frac{FP}{TN+FP}$
- Precision : $P = \frac{TP}{TP+FP}$
- Recall or sensitivity or True Positive Rate (TPR) : $TPR = \frac{TP}{TP+FN}$
- F1score : $F_1 = 2\frac{R*P}{R+P}$
- Training time: the time required by the algorithm to build a data model.
- Testing time: the time required for the classifier to classify new examples.

To avoid false alarms, the FPR should be minimized. The training time and testing time give a good estimate of the performance of the NIDS in terms of speed depending on the size of the dataset. The F1score merges the recall and the precision and its scale varies from 0 to 1. If the recall or the precision is close to 0, the F1score will have a value close to 0 as well. For the F1score to be close to 1, the precision and recall must be high. Particularly, if the dataset is unbalanced, the values of P and R can be misleading. For instance, if the dataset contains 1000 negative instances and 500 positive instances, and the classifier still classifies an instance as positive, the recall will be equal to 1 and the precision will be equal to 0.5 giving the value of $\frac{2}{3}$ for F1 (Balon-Perin, 2012).

4.1.3 System architecture

Figure 26 depicts the NIDS architecture in correlation with the ensemble classifier's proposed methodology. Prior to the training and testing stages, the subset features are defined by using the PCA and Boruta algorithms. These features are then used to train the classifiers (RF, DTs and SVM). Acceptance of variables for each class is established by the Data Pre-processing Unit (DPU). The decision function describes the state of the connection (normal of attack).

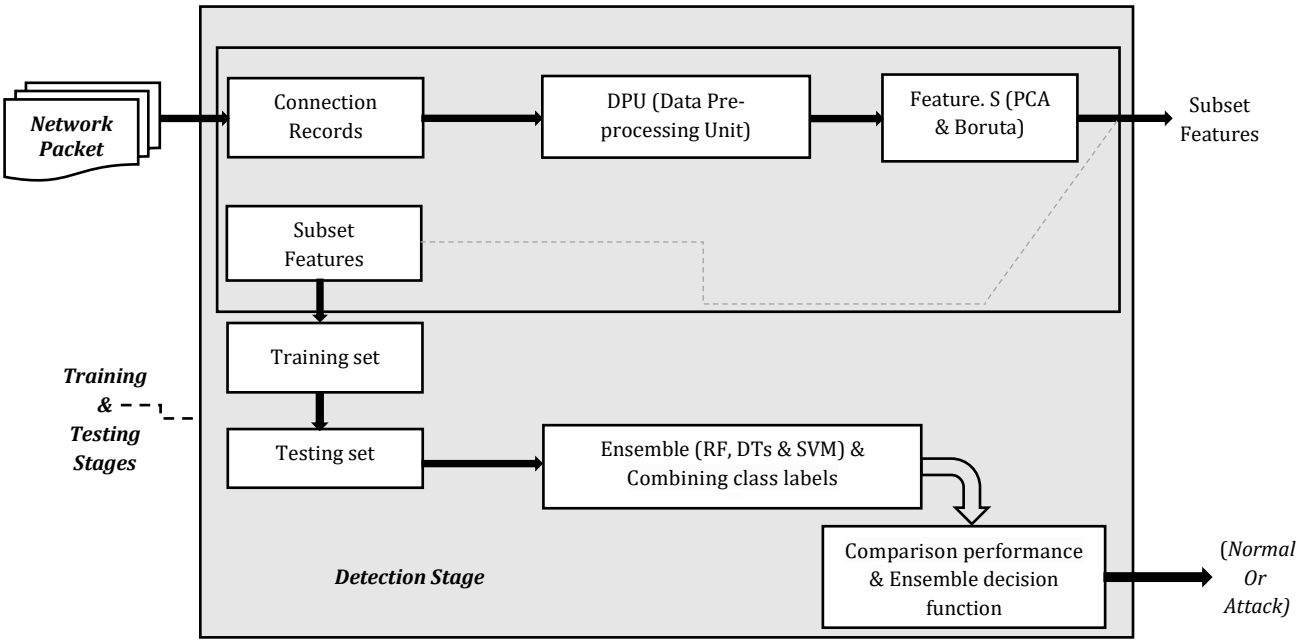


Figure 26 NIDS based-ensemble-Architecture

4.1.4 System tools

The experiments were performed using Intel(R) Core (TM), i5-6500 CPU @3.20 GHz ; 8.00 GB (RAM). 64-bit (OS) and 100.0Mbps (Ethernet). The program used to develop the experiments is RStudio 3.4.4. Initially, the objectives of the experiments are multiple; the experiments should answer the research question stated in chapter 1 by building reliable classifiers. Hence the two targets of the classifiers are:

1. To reduce redundant features and to increase the speed of the NIDS.
2. To use appropriate features on the training, testing stages and final classification in order to increase the overall accuracy.

4.2 Final method

Based on the ensemble strategies, the final method has been summarised into three steps as follows: (1) Input data, DPU and feature selection (2) Ensemble classifiers (3) Decision function**.** The three steps are illustrated in the figure below:

Figure 27 Method of ensemble for the NIDP

Figure 27 illustrates the research method summarised in three steps. The network packet being loaded goes through a DPU. The DPU extracts a number of features from the packet; the features selected by PCA and Boruta algorithms are then sent to the 3 classifiers previously trained to these same features on a training set. The decision function combines the results of each classifier and provides a value describing whether the packet is considered normal or attack. Finally, the intersection of FP and FN sets produced by the 3 classifiers will be evaluated. This will improve the performance that each classifier could achieve.

4.2.1 Ensemble framework

First of all, 3 strategies need to be chosen to build an effective NIDS based-ensemble:

1. Data sampling/selection: making various errors in a given sample is of utmost importance for ensemble-based systems. Moreover, if all sets give the same result, their combination is useless (Polikar, 2012).
2. Training member classifiers: training individual ensemble members using several classifiers (Polikar, 2012).
3. Combining classifiers (combining class labels & combining continuous outputs): the process of combining individual classifiers through different strategies (majority voting, weighted majority voting, mean rule, etc.) (Polikar, 2012).

4.3 Experiments

4.3.1 Overview of the KDD99 dataset

The KDD99 dataset (KDD cup 1999 dataset) was used for the first time in the third international knowledge discovery and data mining tools competition in 1999. It is based on the DARPA98 dataset which was built by the Defence Advanced Research Projects Agency (DARPA) in 1998 during the DARPA98 IDS evaluation program. The DARPA98 dataset includes 7 weeks of data captured as tcpdump from traffic flowing through a network designed for the purposes of the DARPA program. This dataset is divided into training and test sets (Balon-Perin, 2012).

Firstly, the KDD cup 1999 training set contains:

- 4,898,431 records (represented by 41 variables such as duration, src bytes, dst bytes, etc., and a label) and 38 are numerical variables.
- 41 variables (3 are non-numerical): *Protocol type*: 3 protocol types (TCP, UDP and ICMP); *Service*: 70 services (replaced by a number from 1 to 70) and *flag*: 11 flags.

For the 41 features, refer to Appendix A.

Non-numeric variables are transformed into numeric variables to ensure that all ML algorithms will be able to process their values. Figure 28 illustrates the distribution of the examples on the different classes:

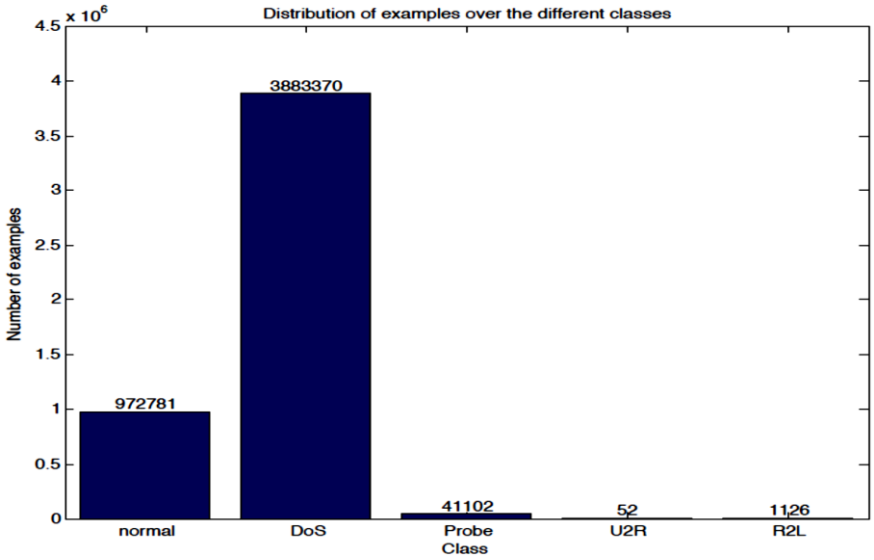


Figure 28 KDD cup 99 training set: Distribution of examples over 5 classes; scale: ×106 (Balon-Perin, 2012)

The training set is greatly unbalanced. Particularly, the classes U2R and R2L are the least well represented with only 52 and 1,126 instances respectively, whereas the DoS class contains 3,883,370 instances. Table 13 shows the distribution of the examples on the different classes; some attacks being marked in red when they count for almost all the examples of the corresponding class.

Table 13 KDD cup 99 training set: Types of attacks in the training set over the different classes (Balon-Perin, 2012)

Probe		U2R		R2L		DoS		Normal	
Attack	**Size**	**Attack**	**Size**	**Attack**	**Size**	**Attack**	**Size**	**Attack**	**Size**
Satan	15,892	buffer_overflow	30	ftp_wrie	8	back	2,203		
portsweep	10,413	loadmodule	9	guess_password	53	land	21		
nmap	2,316	perl	3	imap	12	neptune	1,072,017		
ipsweep	12,481	rootkit	10	multihop	7	pod	264		
				phf	4	smurf	2,807,886		
				spy	2	teardrop	907		
				warezclient	1020				
				warezmaster	20				
Total	41,102	Total	52	Total	1,126	Total	3,883,370	Total	972,781

Secondly, the KDD99 test consists of 311,029 entries. Figure 29 describes the distribution of the examples in the test set on the five classes, which is quite similar to the distribution of the training set.

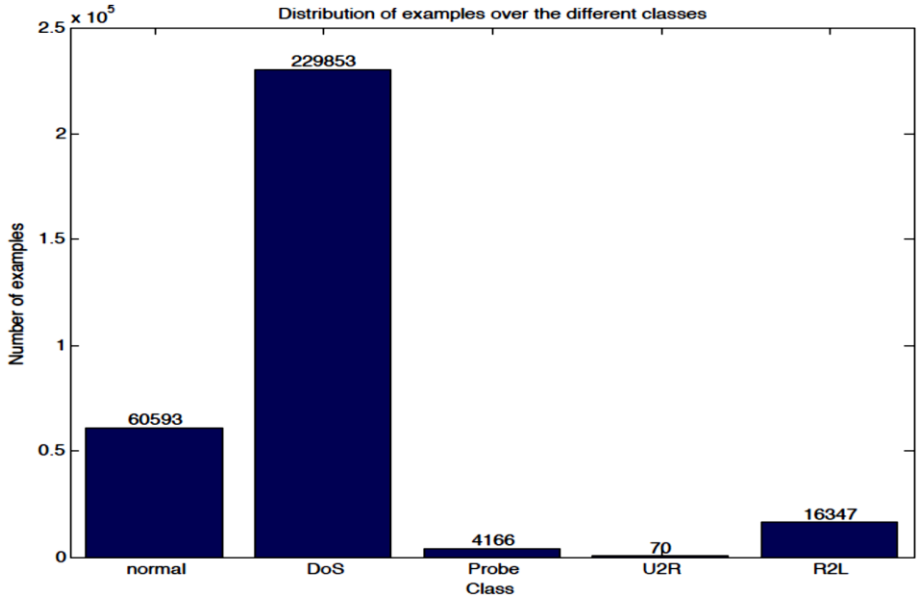


Figure 29 KDD cup 99 test set: Distribution of examples over 5 classes; scale: $\times 10^5$ (Balon-Perin, 2012)

On one hand, the number of instances belonging to the R2L class is ten times greater than the training set. Thus, in order to perform well on the test set, the predictor must have a powerful generalization with 1,126 training examples. On the other hand, "spy" and "warezclient" belonging to the R2L class are not illustrated in the sets. In particular, “warezclient” count for more than 90% of the R2L training set. Finally, two errors were observed in the test set. Records 136,489 and 136,497 have a value for the variable "service" equal to "icmp" which is incorrect. Thus, they were removed from the set before the experiments. However, these errors had already been reported by Tavallaee et al. (2009) in a thorough analysis of the KDD99 dataset (Tavallaee et al., 2009). Table 14 describes the distribution of examples that contain previously unseen attacks and examples that contain attacks already seen for each class:

Table 14 KDD cup 99 test set: Distribution of unseen/seen attacks over the different classes (Balon-Perin, 2012)

Class	Nr of unseen attacks	Nr of seen attacks	Total
Probe	1,789(42.94%)	2,377	4,166
U2R	31(44.29%)	39	70
R2L	10,354(63.34%)	5,993	16,347
DoS	6,555(2.85%)	223,298	229,853
Normal	0	60,593	60,593
Total	18,729(6.02%)	292,300	311,029

The names of the unseen attack types are listed in table 15:

Table 15 KDD cup 99 test set: Unseen attack types (Balon-Perin, 2012)

Class	Unseen attacks types	Total
Probe	mscan,saint	2
U2R	ps,xterm,sqlattack	3
R2L	xlock,snmpgetattack,httptunnel, named,sendmail,snmpguess,worm,xsnoop	8
DoS		
Normal	udpstorm,apache2,mailbomb,processtable	4
Total		17

Tables 14 and 15 show that the number of unseen attacks added in the test set is enormous, especially for the classes U2R, R2L and Probe with 44.29%, 63.34% and 42.94% respectively.

Table 16 indicates the types of attack of the test on the five classes. Some attacks are highlighted in red when they count for almost all the examples in the corresponding class:

Table 16 KDD cup 99 test set: Types of attacks in the test set over the different classes (Balon-Perin, 2012)

Probe		U2R		R2L		DoS		Normal	
Attack	**Size**	**Attack**	**Size**	**Attack**	**Size**	**Attack**	**Size**	**Attack**	**Size**
Satan	1,633	buffer_overflow	22	ftp_wrie	3	back	1,098		
portsweep	354	loadmodule	2	guess_password	4,367	land	9		
nmap	84	perl	2	imap	1	neptune	58,001		
ipsweep	306	rootkit	13	multihop	18	pod	87		
mscan	1,053	ps	16	phf	2	smurf	164,091		
saint	736	sqlattack	2	httptunnel	158	teardrop	12		
		xterm	13	wazermaster	1,602	apache2	794		
				named	17	mailbomb	5000		
				xsnoop	4	processtable	759		
				sendmail	17	udpstorm	2		
				snmpgetattack	7,741				
				snmpguess	2,406				
				worm	2				
				xlock	9				
Total	4,166	Total	70	Total	16,347	Total	229,853	Total	60,593

In addition, a data set as old as the KDD99 cannot cover examples of recent attacks seen within network systems. Hence three major critics of the KDD99 dataset include (Balon-Perin, 2012):

- The unbalanced data distribution (high number of attacks compared to the normal traffic seen in the data set does not reflect the reality of a network in which 99.99 ...% of traffic is normal)
- The redundant records (which introduce bias in the learning phase because of their frequency).
- Variables caught in a controlled environment different from that observed in the wild.

These problems can be solved by sampling appropriate sets of examples in each class of attacks, joining the R2L training set and R2L test set, mixing the result dataset and dividing this new dataset into a new training set and a new one. Therefore the NIDS should be at least accurate on data produced by a simulated environment before being tested on a network where the traffic pattern is unpredictable.

4.3.2 Overview of the NSL-KDD datasets

The NSL-KDD data set (NSL) is a new version of the KDD99 data set. It contains key records from the KDD99 data set (Dhanabal and Shantharajah ,2015). Table 17 shows the list of NSL-KDD files and their descriptions:

Table 17 NSL-KDD Files & Description (Dhanabal and Shantharajah ,2015)

S.N 0.	Name of the file	Description
1	KDDTrain+.ARFF	The full NSL-KDD train set with binary labels in ARFF format
2	KDDTrain+.TXT	The full NSL-KDD train set including attack-type labels and difficulty level in CSV format
3	KDDTrain+_20Percent.ARFF	A 20% subset of the KDDTrain+.arff file
4	KDDTrain+_20Percent.TXT	A 20% subset of the KDDTrain+.txt file
5	KDDTest+.ARFF	The full NSL-KDD test set with binary labels in ARFF format
6	KDDTest+.TXT	The full NSL-KDD test set including attack-type labels and difficulty level in CSV format
7	KDDTest-21.ARFF	A subset of the KDDTest+.arff file which does not include records with difficulty level of 21 out of 21
8	KDDTest-21.TXT	A subset of the KDDTest+.txt file which does not include records with difficulty level of 21 out of 21

The NSL presents the following advantages (Dhanabal and Shantharajah ,2015):

1. Duplicate attributes are removed to allow the classifiers to produce an unbiased result.
2. A sufficient number of records are available in the train and test data sets, which is significant and allows experiments to be performed on the complete set.
3. The number of records selected in each group of difficult levels is inversely proportional to the percentage of records in the original KDD dataset.

The NSL-KDD has 41 attributes with several features attributed to each type (nominal, binary and numeric). Table 18 represents the NSL-KDD attribute value type and features:

Table 18 NSL-KDD_Type & Features (Dhanabal and Shantharajah, 2015)

Type	Features
Nominal	Protocol_type(2), Service(3), Flag(4)
Binary	Land(7), logged_in(12), root_shell(14), su_attempted(15), is_host_login(21), is_guest_login(22)
Numeric	Duration(1), src_bytes(5), dst_bytes(6), wrong_fragment(8), urgent(9), hot(10), num_failed_logins(11), num_compromised(13), num_root(16), num_file_creations(17), num_shells(18), num_access_files(19), num_outbound_cmds(20), count(23) srv_count(24), serror_rate(25), srv_serror_rate(26), rerror_rate(27), srv_rerror_rate(28), same_srv_rate(29) diff_srv_rate(30), srv_diff_host_rate(31), dst_host_count(32), dst_host_srv_count(33), dst_host_same_srv_rate(34), dst_host_diff_srv_rate(35), dst_host_same_src_port_rate(36), dst_host_srv_diff_host_rate(37), dst_host_serror_rate(38), dst_host_srv_serror_rate(39), dst_host_rerror_rate(40), dst_host_srv_rerror_rate(41)

- Nominal : features 2,3 and 4
- Binary: features 7, 12, 14, 15, 21, and 22
- Numeric**:** the rest of the features.

Furthermore, the NSL-KDD includes 5 classes, namely Probe, Normal, DoS, R2L and U2R.

Table 19 shows the distribution of the 5 classes' records in the various NSL-KDD datasets:

Table 19 Distribution of the NSL-KDD_5 classes (Dhanabal and Shantharajah ,2015)

Data Set Type	Total No. of					
	Records	**Normal**	**DoS**	**Probe**	**U2R**	**R2L**
KDD Train+ 20%	25192	13449	9234	2289	11	209
		53.39%	36.65%	9.09%	0.04%	0.83%
KDD Train+	125973	67343	45927	11656	52	995
		53.46%	36.46%	9.25%	0.04%	0.79%
KDD Test+	22544	9711	7458	2421	200	2754
		43.08%	33.08%	10.74%	0.89%	12.22%

The attack classes in the NSL-KDD dataset are further subdivided into four categories. Table 20 describes the 41 attributes associated with different attack type:

Table 20 NSL-KDD_Attack class & Attack type (Dhanabal and Shantharajah, 2015)

Attack class	Attack Type
DoS	Back, Land, Neptune, Pod, Smurf,Teardrop,Apache2, Udpstorm, Processtable, Worm (10)
Probe	Satan, Ipsweep, Nmap, Portsweep, Mscan, Saint (6)
R2L	Guess_Password, Ftp_write, Imap, Phf, Multihop, Warezmaster, Warezclient, Spy, Xlock, Xsnoop, Snmpguess, Snmpgetattack, Httptunnel, Sendmail, Named (16)
U2R	Buffer_overflow, Loadmodule, Rootkit, Perl, Sqlattack, Xterm, Ps (7)

The NSL-KDD dataset suggested solving some of the problems inherent in the KDD99 dataset. However, KDD99 remains the most widely used data set for IDS. NSL-KDD includes selected records from the full KDD, which solve some problems such as unbalanced distribution of data and so forth (Tavallaee et al., 2009).

4.3.3 Experiment procedures

4.3.3.1 Description of the procedures

Firstly, each problem (normal, probe, U2R, R2L, and DoS) is examined by one of the three classifiers (RF, DTs and SVM). This will enable to analyze each problem individually in the experiments and merge the sub-solutions into a global solution. The data sets are divided into five sections, with one section for each class of attacks and for normal instances. In addition, no classifier was initially built to detect normal instances. To have a balanced dataset of abnormal instances and normal instances, a dataset must be created for each problem by selecting a number of attributes in the attack class and the normal class.

Secondly, for the few instance classes (U2R and R2L), the complete set is selected, giving a U2R training set of 104 examples (52 U2R and 52 Normal) and a set of R2L training of 2,252 instances (1,126 R2L and 1,126 Normal). Afterward, the labels of the data sets for each class of attacks are modified to add a computational overhead and allow binary classifiers such as RF to process the data. The two labels are 0 for a normal instance and 1 for an anomalous instance. This would enhance the accuracy of the alert message sent by the NIDS to the network administrator.

Lastly, the equation given in section 4.1.2 will be used to measure and evaluate the performance of the NIDS. The experiments carried out in this work are in direct continuity with the works of Zainal et al. (2009), Balon-Perin (2012) and Mukkamala et al. (2005). In particular, Balon-Perin and Srinivas Mukkamala identified features related to each class using different feature selection techniques.

4.3.3.2 Experimentation Stages

The experimentations comprise two stages:

1. Training stage: the system builds a model using the training data to give maximum generalization precision on unseen and undetected variables.
2. Testing stage: the data is tested with the model built to detect and identify possible intrusions in the training stage.

4.4 Assessments

4.4.1 Feature selection assessment

The feature selection is a very effective way of reducing the size of a problem. Redundant and duplicated attributes are removed from the data before being sent to the ML classifiers. Feature selection is an independent pre-processing phase of the learning algorithm. Generally, automatic feature selection works much better than manually because the algorithm is able to find obvious correlations between different features (Balon-Perin and Gambäck, 2013). The feature selection techniques applied in this work have sensibly reduced the number of attributes of the KDD99 and NSL datasets.

The purpose of evaluating the selected attributes is not to find the best algorithm to adjust them accurately. Instead, the goal is to conclude on the effectiveness of the technique with a smaller set of features. Three benefits of feature selection (Balon-Perin and Gambäck, 2013):

1. Improves the computational speed with minimal reduction of accuracy.
2. Reduces noise and robust against over-fitting.
3. Uses when the problem has a high dimensionality.

The main feature selection algorithms applied in this dissertation are the principal component analysis (PCA) and Boruta algorithms. These algorithms aim at extracting as much information as possible from the smallest possible number of attributes (Balon-Perin and Gambäck, 2013). Firstly, after applying the PCA algorithm and removing the duplicate attributes from the KDD99 datasets, DoS contains the most observations (391458 observations). Table 21 details the new dataset variables:

Table 21 PCA algorithm on the KDD99

Dos:	391458
Normal:	97278
Probe:	4107
R2L:	1126
U2R:	52

Afterwards, within +1.2 hours Boruta performed 12 iterations on the KDD99 where:

- 36 attributes confirmed important: count, diff_srv_rate, dst_bytes, dst_host_count, dst_host_diff_srv_rate and 31 more.
- 2 attributes rejected: is_hot_login, num_outbound_cmds; still have 2 attributes left.

Within 2.30092 hours, Boruta performed 24 iterations where:

- 36 attributes confirmed important: count, diff_srv_rate, dst_bytes, dst_host_count, dst_host_diff_srv_rate and 31 more.
- 3 attributes confirmed unimportant: is_hot_login, num_outbound_cmds, urgent.
- 1 tentative attributes left: su_attempted.

Secondly, after applying the PCA algorithm and removing the duplicate attributes from the NSL datasets, normal shows the most observations (67343).Table 22 details the new dataset variables:

Table 22 PCA algorithm on the NSL KDD

normal:	67343
neptune :	41214
satan :	3633
ipsweep :	3599
portsweep :	2931
smurf :	2646
(Other) :	4607

Within 47.83339 mins (48mins), Boruta performed 20 iterations on the NSL where:

- 38 attributes confirmed important: `NA`, count, diff_srv_rate, dst_bytes, dst_host_count and 33 more.
- 3 attributes confirmed unimportant: is_hot_login, num_outbound_cmds, urgent.

4.4.2 Test set assessment

The KDD99 datasets have been split into two files “training” with 49404 observations of 26 variables and “testing” with 444617 observations of 28 variables. The NSL datasets have also been split into “training” (12608 observations of 26 variables) and “testing” (113365 observations of 28 variables) files.

4.5 Summary

The purpose of this chapter was to build the NIDS based-ensemble and to study the feasibility of such a system by determining whether its implementation is justified or not. To achieve this, it was necessary to analyse the reliability of the framework in terms of feature selection and test assessments. This chapter reviewed the ensemble method and strategies, system requirements as well as the dataset used during the training and testing stages. The next chapter (chapter 5), analyses and discusses the overall results.

Chapter 5-Results and Discussion

The three great essentials to achieve anything worthwhile are: Hard work, Stick-to-itiveness, and Common sense…

-Thomas A. Edison-

This chapter presents and discusses the findings of the study. RStudio was used as a support and tool of implementation for the NIDS. Then to fulfil the stated requirements (in chapter 4), the system has been tested on the KDD99 and NSL KDD datasets which provided a representative basis for the NIDS evaluation.

5.1 Exploratory Analysis

5.1.1 On KDD99

Figure 30 illustrates the percentages (%) of connections to current host having same src (source) port (dst_host_same_src_port_rate) and the % of connections to same service coming from different hosts (dst_host_srv_diff_host_rate). Dst_host_same_src_port_rate shows slight effect on the intrusion type while dst_host_srv_diff_host_rate gives value greater than equal to 1 for probe and R2L:

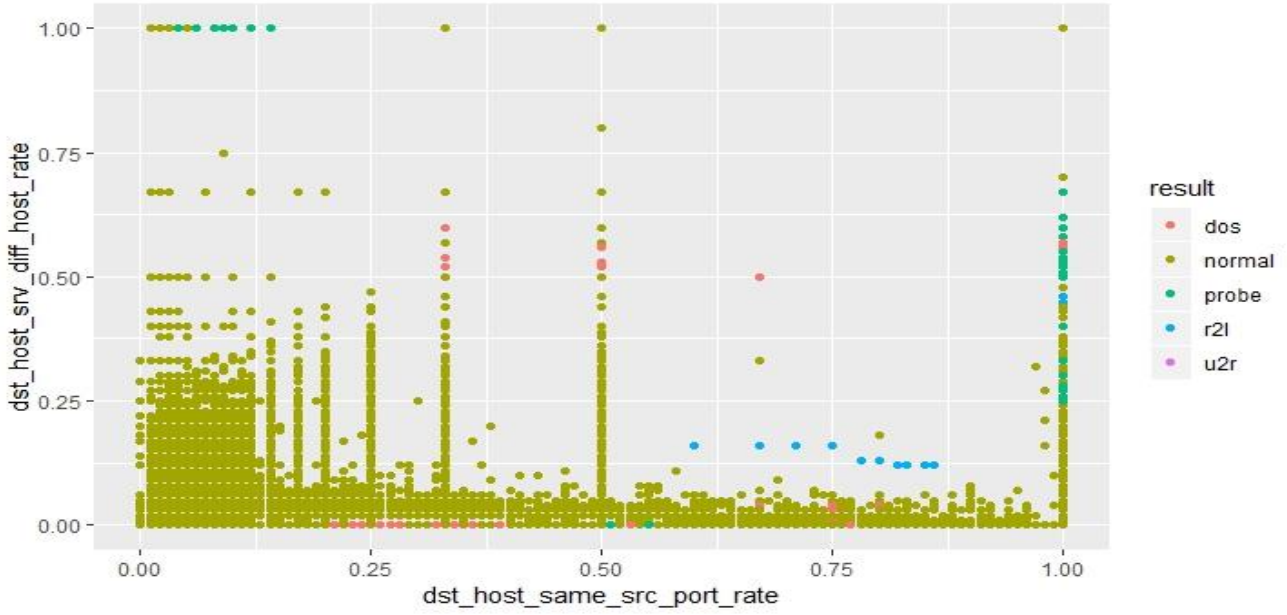


Figure 30 Same src port vs different hosts

Figure 31 shows the percentages (%) of the duration of connection in seconds which is greater than 30000s and the number of data bytes from src to dst (destination) port (src_bytes). Duration seems to be a great predictor for probe class:

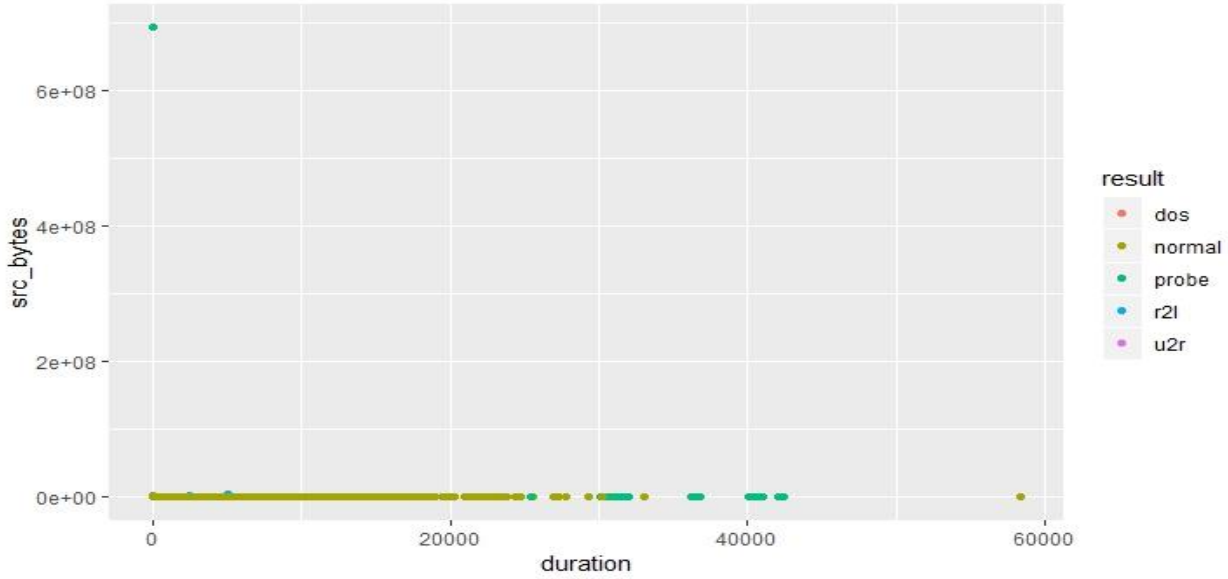


Figure 31 Duration of connection vs number of data bytes from source to destination

Figure 32 illustrates various instances of normal class or error status flag of connection ("REJ" and "S0" = "DoS") and the destination port mapped to service. Flag is a strong predictor for DoS class:

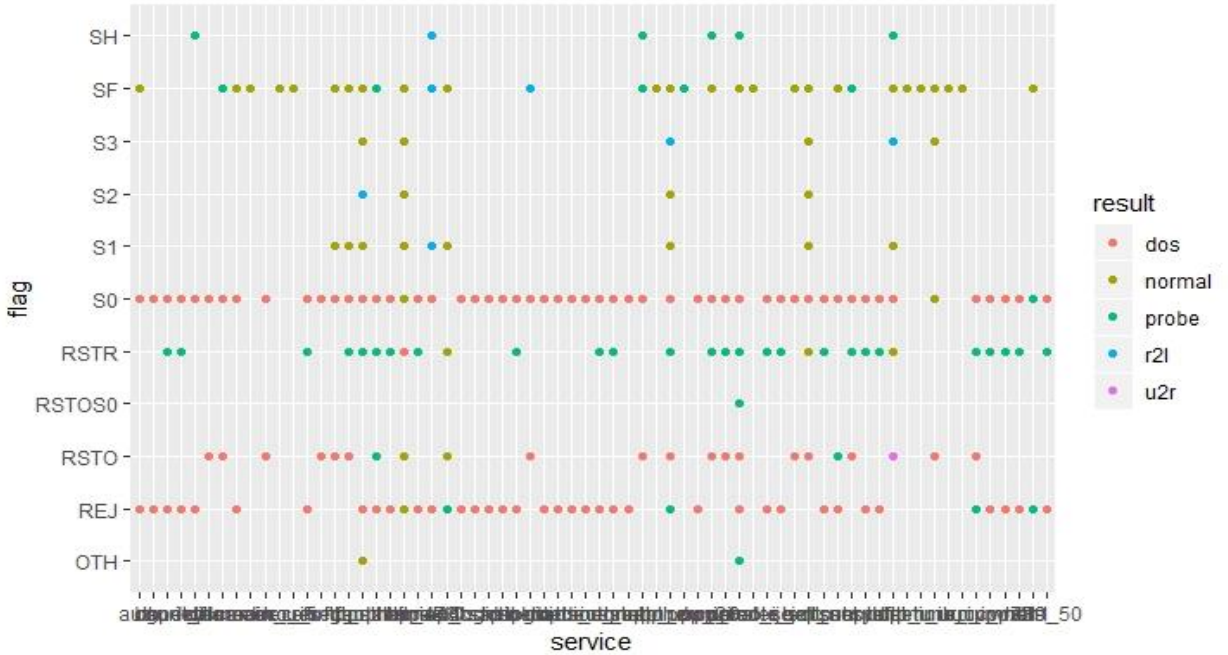


Figure 32 Service vs flag -normal or error status flag of connection

5.1.2 On NSL

Figure 33 shows various instances of service on neptune, pod, teardrop and smurf and the protocole_type (tcp,udp,icmp) seems to be a strong predictor for "DoS":

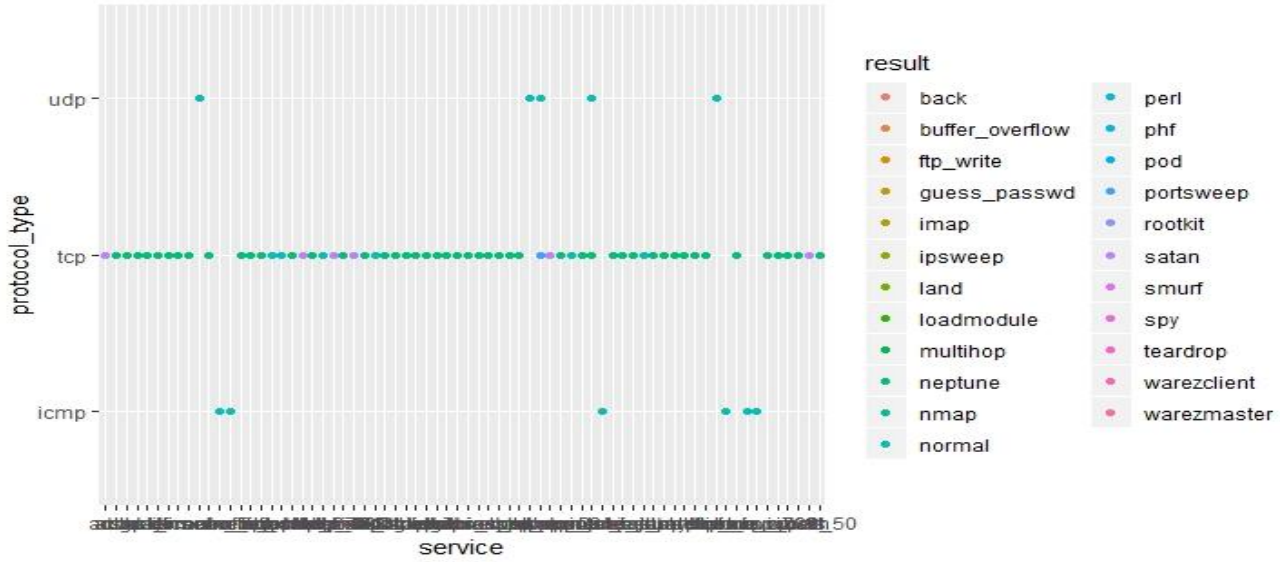


Figure 33 Service vs protocol_type

Figure 34 shows the % of connections that have "SYN[3]" errors (srv_serror_rate) (0.25-0.5 equal to probe type) and the % of connections that have "SYN" errors (serror_rate) (0 or 1 equal to DoS type):

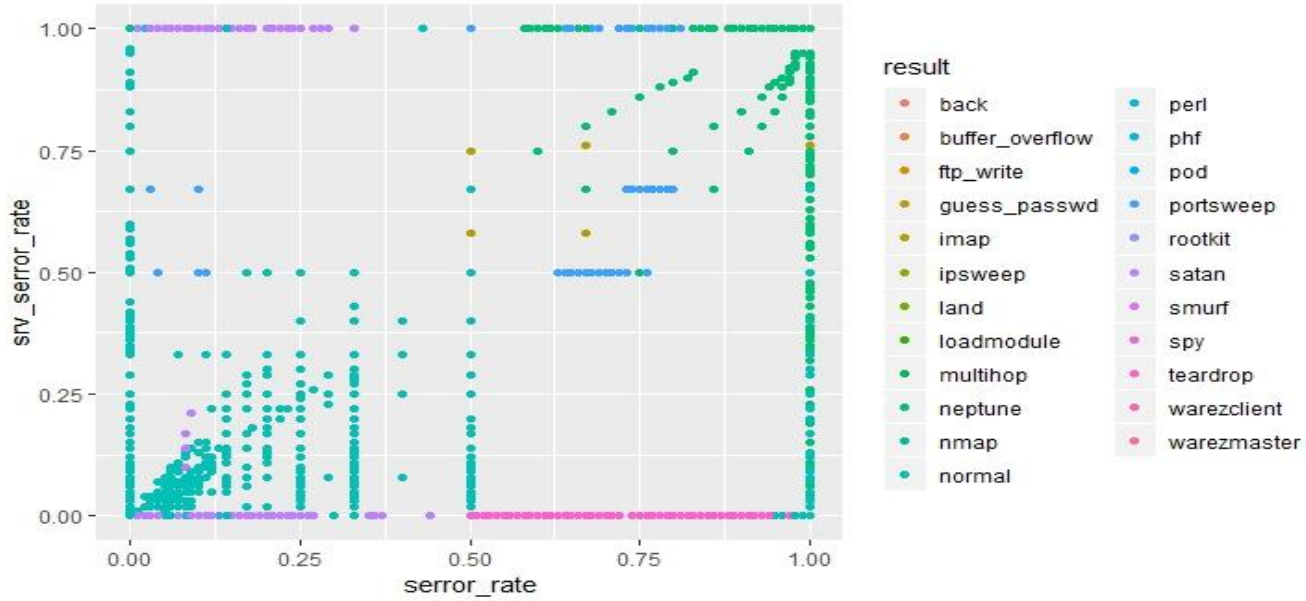


Figure 34 Connections with "SYN" errors vs service record

[3] Also referred to as the DoS class-SYN/ACK (Synchronization/Acknowledged)

5.2 Results Analysis

This section shows the predictions and models of each classifier based on:

5.2.1 The KDD99

5.2.1.1 RF

Table 23 depicts the RF prediction where "DoS" type contains the most observations (352221) among the five classes considered in this prediction. Variables identified by the same class are highlighted as they are specific to RF based on the KDD99 datasets.

Table 23 RF-Prediction on the KDD99

RF-Pred.	Dos	Normal	Probe	R2L	U2R
Dos	352221	8	5	5	4
Normal	79	87527	53	125	37
Probe	12	4	3616	0	0
R2L	0	10	0	881	0
U2R	0	1	0	2	5

Figure 35 shows the RF model with the number of connections to same host as current connection in past two seconds; "count" variable is the most important element (>2000) among the attributes considered in the RF model.

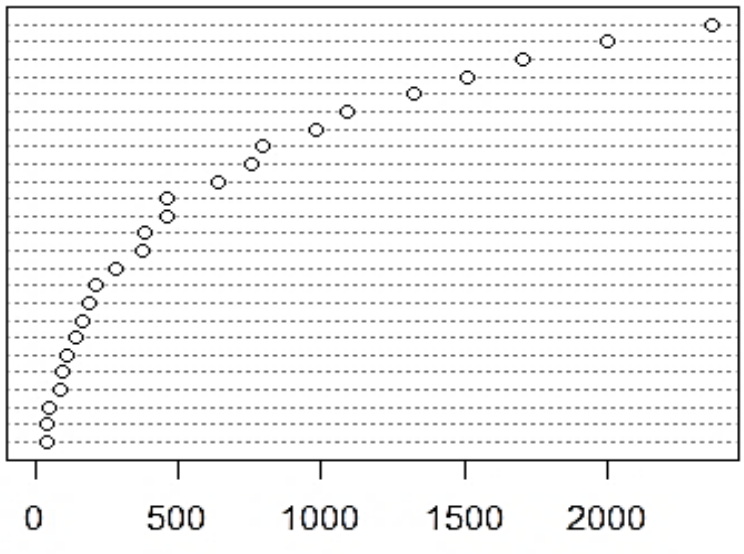


Figure 35 RF-model-1 on various features

Figure 36 illustrates the ROC-AUC with the separability measure (between the TPR or sensitivity vs the FPR or specificity) equals to 1. Particularly, this model shows various TPRs (0.0 - 0.8) vs FPRs (1.0 – 0.0) classified as abnormal. Hence there is 100% probability that the RF model will be able to demarcate attacks from normal class. This is quite satisfactorily.

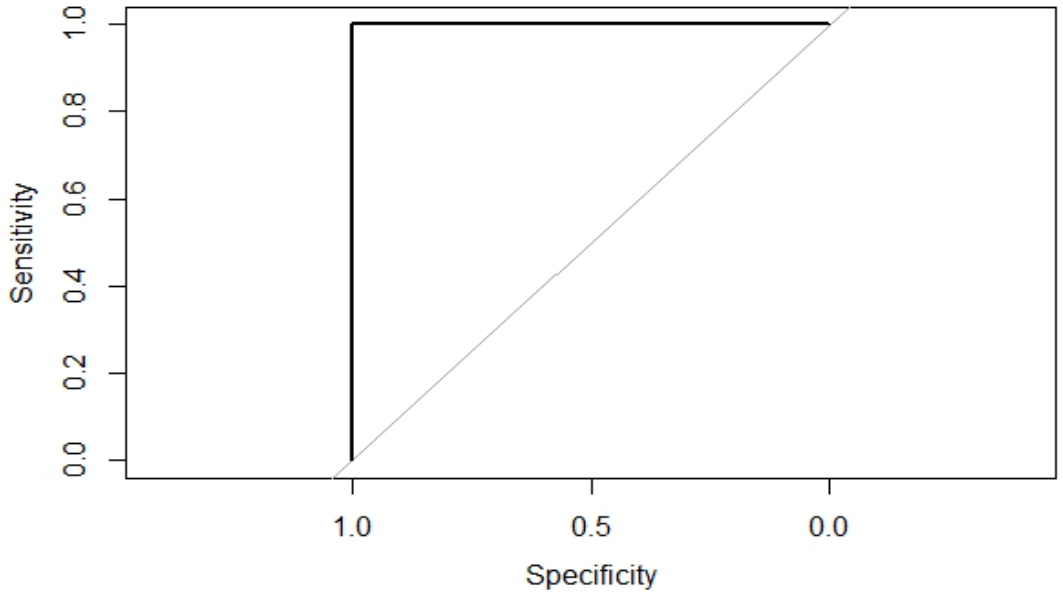


Figure 36 RF-model (ROC - AUC)

5.2.1.1 DTs

Table 24 shows the DTs prediction where "DoS" type contains the most observations (351536) among the five classes considered in this prediction. Variables identified by the same class are highlighted as they are specific to DTs based on the KDD99 datasets.

Table 24 DTs-Prediction on the KDD99

DTs-Pred.	Dos	Normal	Probe	R2L	U2R
Dos	351536	193	786	64	0
Normal	687	87249	561	929	42
Probe	89	108	2349	20	4
R2L	0	0	0	0	0
U2R	0	0	0	0	0

Figure 37 illustrates the DTs classifier based on the subset selected by PCA and Boruta techniques including where DoS contains the most observations of the training phase (39146); including dst_bytes<2 , diff_srv_rate<0.24, count >=55, src_bytes, flag and dst_host_srv_diff_host_rate.

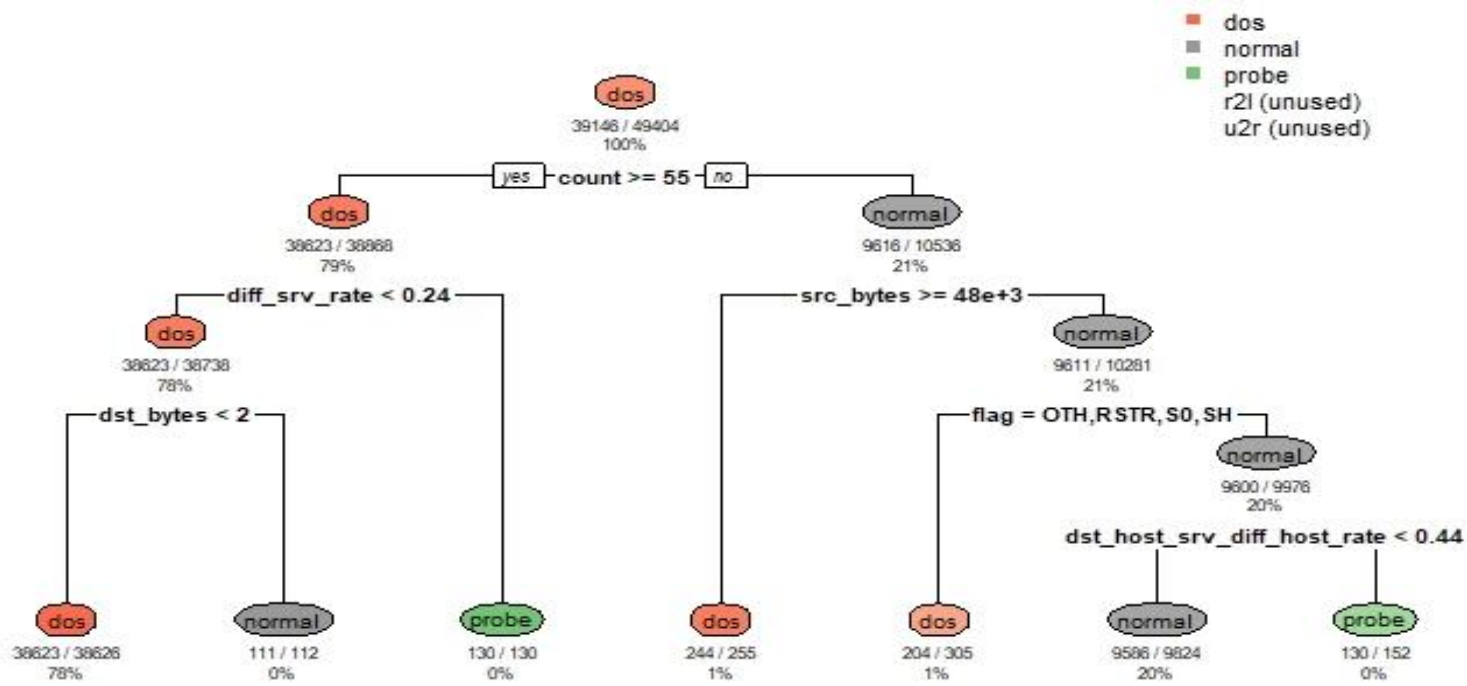


Figure 37 DTs-model

5.2.1.1 SVM

Table 25 shows the SVM prediction where "DoS" type contains the most observations (351409) among the five classes considered in this prediction. Variables identified by the same class are highlighted as they are specific to SVM based on the KDD99 datasets.

Table 25 SVM-Prediction on the KDD99

SVM-Pred.	**Dos**	**Normal**	**Probe**	**R2L**	**U2R**
Dos	351409	100	48	18	14
Normal	827	87344	101	204	29
Probe	72	31	3542	4	1
R2L	4	75	5	784	2
U2R	0	0	0	0	0

5.2.2 The NSL

Figure 38 illustrates the RF model with "src_bytes" variable as the most important element (>=1200) of this model showing the number of data bytes from source to destination. The rest of the variables show the duration, the % of connections that have "SYN" errors, the protocole type and the number of connections having same destination host, using same service and current connection of 4Mbps.

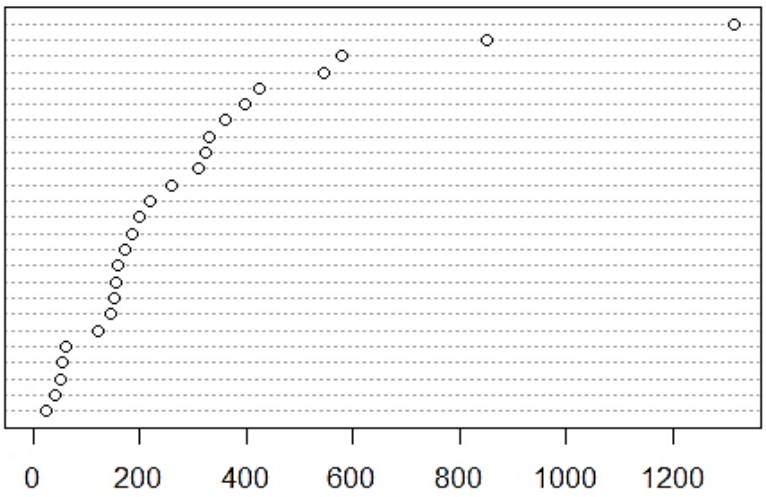


Figure 38 RF-model-2 on NSL

Figure 39 illustrates the Mtry[4] on the NSL datasets with:

- Mtry = 5 OOB error = 0.09%
- Mtry = 6 OOB error = 0.08%
- Mtry = 7 OOB error = 0.08%

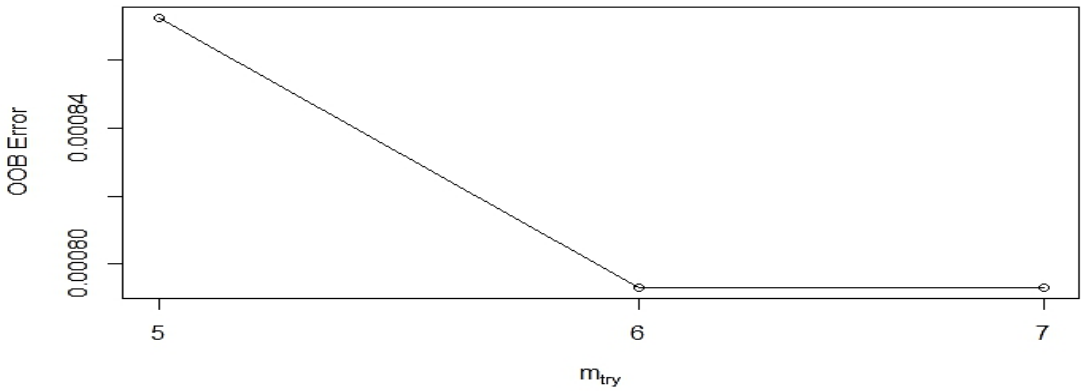


Figure 39 Number of predictors-Mtry

[4] Number of predictors sampled for splitting at each node in the RF (Breiman,2001)

5.3 Comparison of the results

The three classifiers (RF, DTs and SVM) will be compared to each other in terms of prediction and accuracy. The confusion matrix is being used to compare the performance and detection rate of each classifier.

5.3.1 The KDD99

This section shows the performance comparison of each classifier based on the KDD99 datasets:

5.3.1.1 RF

Table 26 shows the RF results based on the PCA and Boruta techniques with the accuracy metric of 0.9991746 (99%). "DoS" class shows the highest percentage among the five classes considered in this model (99.99%).

Table 26 RF-Confusion matrix-1

	Dos	Normal	Probe	R2L	U2R
Dos	99.99	0.00	0.01	0.00	0.00
Normal	0.09	99.67	0.06	0.14	0.04
Probe	0.33	0.11	99.56	0.00	0.00
R2L	0.00	1.12	0.00	98.88	0.00
U2R	0.00	12.50	0.00	25.00	62.50

5.3.1.2 DTs

Table 27 shows the DTs results based on the PCA and Boruta techniques with the accuracy metric of 0.9921663 (99%). "DoS" class shows the highest percentage among the five classes considered in this model (99.70%).

Table 27 DTs-Confusion matrix-1

	Dos	Normal	Probe	R2L	U2R
Dos	99.70	0.05	0.22	0.02	0
Normal	0.77	97.52	0.63	1.04	0.05
Probe	3.46	4.20	91.40	0.78	0.16
R2L	0.00	0.00	0.00	0.00	0.00
U2R	0.00	0.00	0.00	0.00	0.00

5.3.1.3 SVM

Table 28 shows the SVM result based on the PCA and Boruta techniques with the accuracy metric of 0.9965476 (99%). "DoS" class shows the highest percentage among the five classes considered in this model (99.95%).

Table 28 SVM-Confusion matrix-1

	Dos	Normal	Probe	R2L	U2R
Dos	99.95	0.03	0.01	0.01	0
Normal	0.93	98.69	0.11	0.23	0.03
Probe	1.97	0.85	97.04	0.11	0.03
R2L	0.46	8.59	0.57	90.15	0.23
U2R	0.00	0.00	0.00	0.00	0.00

Table 29 shows the overall result of the three classifiers based on the KDD99 datasets subdivided into 5 different classes (DoS, normal, probe, R2L, U2L):

Table 29 Ensemble classifier-1

Classes	**RF**	**DTs**	**SVM**
	Accuracy	**Accuracy**	**Accuracy**
DoS	99.99	99.7	99.95
Normal	99.67	97.52	98.69
Probe	99.56	91.4	97.04
R2L	98.88	0.0	90.15
U2R	62.5	0.0	0.0

From Table 29, RF does have significantly higher accuracy than DTs and SVM. The predictive accuracy of SVM is slightly higher than that of DTs. The evaluation results show the accuracy metric between 99.7% and 99.95%, which is quite optimal and acceptable compared to the works conducted by Balon-Perin (2012) and Zainal et al. (2009). The results in table 29 demonstrate that using RF for the NIDP is a viable proposition. RF classifier outperformed SVM and DTs in prediction and accuracy.

5.3.2 The NSL

This section shows the performance comparison of each classifier based on the NSL data sets:

5.3.2.1 RF

Table 30 shows the RF result based on the PCA and Boruta techniques with the accuracy metric of 0.9913377 (99%) among the attributes selected in this prediction. Features having the highest % and identified by the same class are highlighted as they are specific to RF based on the NSL datasets. "Warezmaster" which is a type of R2L, shows the highest percentage among the 23 features considered in this model (100%).

Table 30 RF-Confusion matrix-2

	smurf	spy	teardrop	warezclient	warezmaster
back	0.0	0.0	0.0	0.0	0.0
buffer_overflow	0.0	0.0	0.0	0.0	0.0
ftp_write	0.0	0.0	0.0	0.0	0.0
guess_passwd	0.0	0.0	0.0	0.0	0.0
imap	0.0	0.0	0.0	0.0	0.0
ipsweep	0.03	0.0	0.0	0.0	0.0
land	0.0	0.0	0.0	0.0	0.0
loadmodule	0.0	0.0	0.0	0.0	0.0
multihop	0.0	0.0	0.0	0.0	0.0
neptune	0.0	0.0	0.0	0.0	0.0
nmap	0.0	0.0	0.0	0.0	0.0
normal	0.02	0.0	0.05	0.16	0.03
perl	0.0	0.0	0.0	0.0	0.0
phf	0.0	0.0	0.0	0.0	0.0
pod	3.01	0.0	0.0	0.0	0.0
portsweep	0.0	0.0	0.0	0.0	0.0
rootkit	0.0	0.0	0.0	0.0	0.0
satan	0.0	0.0	0.0	0.0	0.0
smurf	99.92	0.0	0.0	0.0	0.0
spy	0.0	0.0	0.0	0.0	0.0
teardrop	0.0	0.0	99.87	0.0	0.0
warezclient	0.0	0.0	0.0	96.97	0.0
warezmaster	0.0	0.0	0.0	0.0	100

5.3.2.2 DTs

Table 31 shows the DTs result based on the PCA and Boruta techniques with the accuracy metric of 0.9536453 (95%) among the attributes selected in this prediction. Features having the highest % and identified by the same class are highlighted as they are specific to DTs based on the NSL datasets. "smurf" which is a type of DoS, shows the highest percentage among the 23 features considered in this model (98.65%).

Table 31 DTs-Confusion matrix-2

	rootkit	satan	smurf	spy	teardrop	warezclient	warezmaster
back	0.0	0.0	0.0	0.0	0.0	0.0	0.0
buffer_overflow	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ftp_write	0.0	0.0	0.0	0.0	0.0	0.0	0.0
guess_passwd	0.0	0.0	0.0	0.0	0.0	0.0	0.0
imap	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ipsweep	0.0	0.0	0.0	0.0	0.0	0.0	0.0
land	0.0	0.0	0.0	0.0	0.0	0.0	0.0
loadmodule	0.0	0.0	0.0	0.0	0.0	0.0	0.0
multihop	0.0	0.0	0.0	0.0	0.0	0.0	0.0
neptune	0.0	1.10	0.0	0.0	0.0	0.0	0.0
nmap	0.0	0.0	0.0	0.0	0.0	0.0	0.0
normal	0.01	0.16	0.06	0.0	1.26	1.25	0.03
perl	0.0	0.0	0.0	0.0	0.0	0.0	0.0
phf	0.0	0.0	0.0	0.0	0.0	0.0	0.0
pod	0.0	0.0	0.0	0.0	0.0	0.0	0.0
portsweep	0.0	3.68	0.0	0.0	0.0	0.0	0.0
rootkit	0.0	0.0	0.0	0.0	0.0	0.0	0.0
satan	0.0	87.42	0.0	0.0	0.0	0.0	0.0
smurf	0.0	0.0	98.65	0.0	0.0	0.0	0.0
spy	0.0	0.0	0.0	0.0	0.0	0.0	0.0
teardrop	0.0	0.0	0.0	0.0	0.0	0.0	0.0
warezclient	0.0	0.0	0.0	0.0	0.0	0.0	0.0
warezmaster	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5.3.2.3 SVM

Table 32 shows the RF result based on the PCA and Boruta techniques with the accuracy metric of 0.9824373(98%) among the attributes selected for this model. Features having the highest % and identified by the same class are highlighted as they are specific to SVM based on the NSL datasets. "Warezmaster" which is a type of R2L, shows the highest percentage among the 23 features considered in this model (100%).

Table 32 SVM-Confusion matrix-2

	smurf	spy	teardrop	warezclient	warezmaster
back	0.0	0.0	0.0	0.0	0.0
buffer_overflow	0.0	0.0	0.0	0.0	0.0
ftp_write	0.0	0.0	0.0	0.0	0.0
guess_passwd	0.0	0.0	0.0	0.0	0.0
imap	0.0	0.0	0.0	0.0	0.0
ipsweep	0.0	0.0	0.0	0.0	0.0
land	0.0	0.0	0.0	0.0	0.0
loadmodule	0.0	0.0	0.0	0.0	0.0
multihop	0.0	0.0	0.0	0.0	0.0
neptune	0.0	0.0	0.0	0.0	0.0
nmap	0.0	0.0	0.0	0.0	0.0
normal	0.16	0.0	0.22	0.33	0.01
perl	0.0	0.0	0.0	0.0	0.0
phf	0.0	0.0	0.0	0.0	0.0
pod	0.0	0.0	0.0	0.0	0.0
portsweep	0.0	0.0	0.0	0.0	0.0
rootkit	0.0	0.0	0.0	0.0	0.0
satan	0.0	0.0	0.0	0.0	0.0
smurf	97.03	0.0	0.0	0.0	0.0
spy	0.0	0.0	0.0	0.0	0.0
teardrop	0.0	0.0	95.89	0.0	0.0
warezclient	0.0	0.0	0.0	81.73	0.0
warezmaster	0.0	0.0	0.0	0.0	100

Table 33 shows the overall result of the three classifiers based on the NSL KDD subdivided into 5 different classes:

Table 33 Ensemble classifier-2

Classes	**RF**	**DTs**	**SVM**
	Accuracy	**Accuracy**	**Accuracy**
DoS	99.92	98.65	97.03
Normal	0.02	0.16	0.0
Probe	0.0	87.42	0.0
R2L	96.97	0.0	100
U2R	0.0	0.0	0.0

From Table 33, RF does have significantly higher accuracy than DTs and SVM. The predictive accuracy of SVM is slightly higher than that of DTs. The predictions and evaluations show the accuracy metric between 95% and 99%, which is quite satisfactorily and acceptable compared to the works conducted by Balon-Perin (2012) and Zainal et al. (2009). Table 34 illustrates false positives rate (FPR) or false alarms rate (FAR). FPR is obtained by the following equation:

$$FPR = \frac{FP}{TN+FP}$$

Prior to final ensemble model, FPR is used to assess the performance of the system. Different classes (probe, R2L and U2R) have been selected to evaluate the prediction of each classifier. 72%, 89.02% and 12.03% are considered as probe; which means that an attacker can easily have access the active machines and ports within the network system. 4.01% and 0% are considered to be R2L; the attacker cannot exploit the vulnerability of the system. 0% is considered to be U2R; confidential information will be protected. To avoid false positives or alarms, the FPR should be minimized. Hence this shows a complete minimization of FPs and FAs.

Table 34 FPR based on individual classifier

	FPR		
Pred.	**SVM**	**DTs**	**RF**
Probe	72.02	89.02	12.03
R2L	4.01	0	0
U2R	0	0	0

Table 35 depicts the final ensemble model by combining appropriate features of the five classes (DoS, Normal, Probe, R2R and U2R). SVM has the highest accuracy on the R2R class (100%).

Table 35 Final ensemble based on accuracy rate

Classes	**Ensemble class -1 (KDD99)**	**Classes**	**Ensemble class-2 (NSL KDD)**	**Final Ensemble**
	Accuracy		**Accuracy**	**Accuracy**
DoS	99.99	**DoS**	99.92	99.99
Normal	99.67	**Normal**	0.16	99.67
Probe	99.56	**Probe**	87.42	99.56
R2L	98.88	**R2L**	100	100
U2R	62.5	**U2R**	0.0	62.5

Figure 40 shows the percentages of each class based on the KDD99 and NSL datasets. The performance of three classifiers have brought substantial improvement on DoS class with 22%, normal class with 22%, probe class with 21%, R2L class with 22%, U2R class with 13%. In general, RF, DTs and SVM block 87.5% of intrusions and attempts to write on any server. Hence the intruder cannot upload unauthorized files on any server nor download illegal files during an attack attempt.

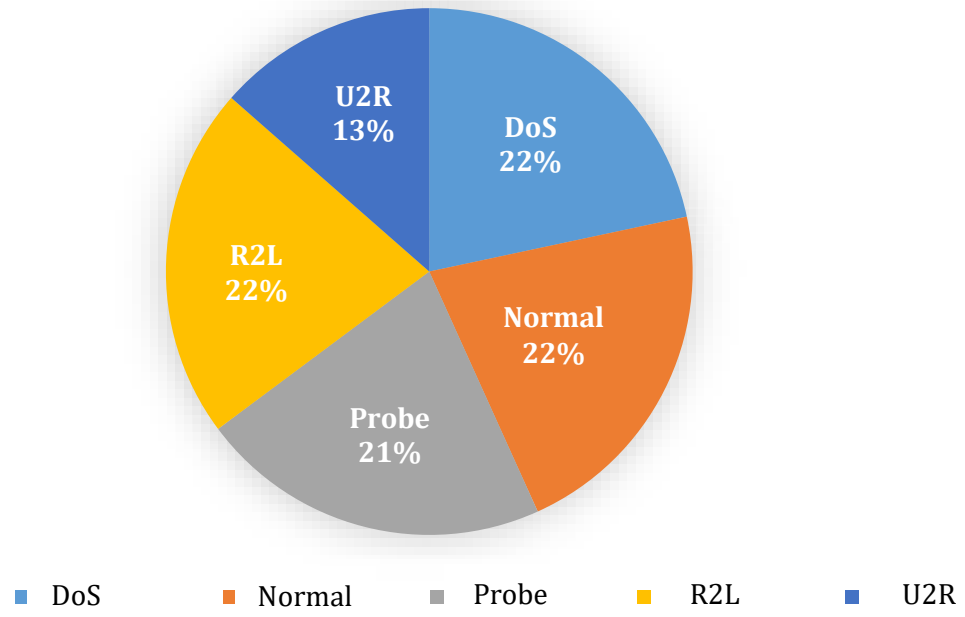


Figure 40 Percentages (%) of the 5 classes

Figure 41 shows the average accuracy of the three classifiers on the 5 classes of attack. On average, 92.34% of the DoS intrusions are identified. It is thus impossible for an attacker to deny access to a resource to one or more legitimate users of a network. The entries or addresses related to a gateway in the ARP tables cannot be modified or altered by an intruder. 90.43% of normal connection or normal host session on the network system. 87.35% of the probe attacks are detected. Hence it is impossible for an attacker to have access to a remote machine or server within the network. The attacker will not have a complete map of the active machines and ports within the network system. In general, the intruder will also have hard time finding the vulnerabilities in the network, the type of the server and version of the OS. 81.25% of R2L are identified. It is then difficult for an attacker to log-in from a computer located outside the targeted network, to exploit the vulnerability of the system and to access a computer on the LAN.

Typically, the attacker cannot manipulate or use a fake temporary file within the network system. 62.5% of U2R are detected. Thus, the attacker cannot log-on to a computer within the network. The NIDS deprives the intruder from exploiting some vulnerability of the program installed on the computer and executing a command from any root shell; the user cannot easily access the confidential information. The attacker will have hard time controlling the host computer and server, launching a virus and finding the correct code to run a root shell within the network connection records or host session and program's memory.

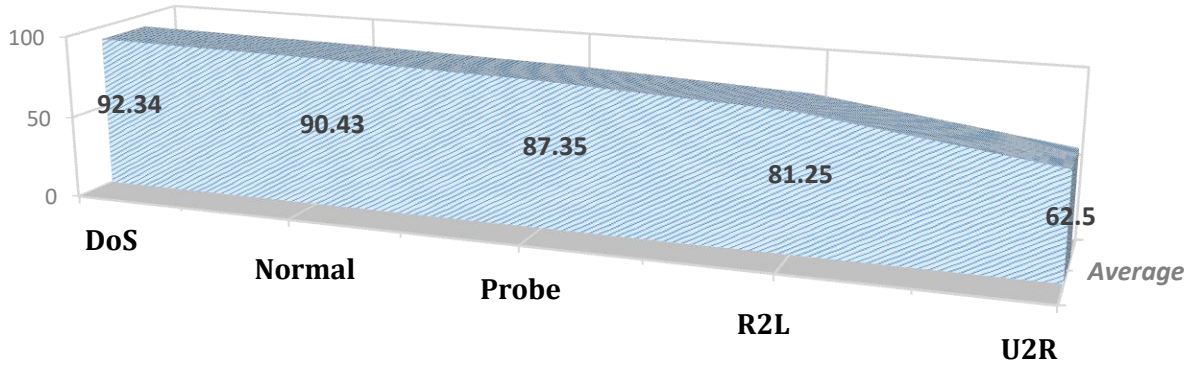


Figure 41 Average Accuracy (%)

Furthermore, ensemble techniques can be implemented to protect vulnerable systems and servers. Particularly, DTs enable systems to identify suspicious and malicious traffic sources. SVM detects malware and attacks within servers, personal network and LAN. RF identifies anomalies, DDoS attacks and unauthorised devices within a system (Al-Garadi et al., 2018). In general, the combinatorial approach creates a robust model (NIDS based-ensemble techniques) by improving the overall results of each technique. These techniques are sufficiently streamlined to provide in-line protection for critical infrastructure by compromising the following aspects:

- Authentication (information comes from a verifiable source).
- Confidentiality (protection of confidential information).
- Integrity (no alteration of information).
- Availability (the system remains operable).

Figure 42 shows the final ensemble performances based on accuracy rate of each classifier. The comparison results indicates that SVM performs best (100%) on R2L class. RF performs well (99.99%) on DoS class. DT performs quite well (98.65%) on DoS class. Particularly, the application of SVM has a larger improvement than RF and DTs in terms of accuracy rate on R2L class.

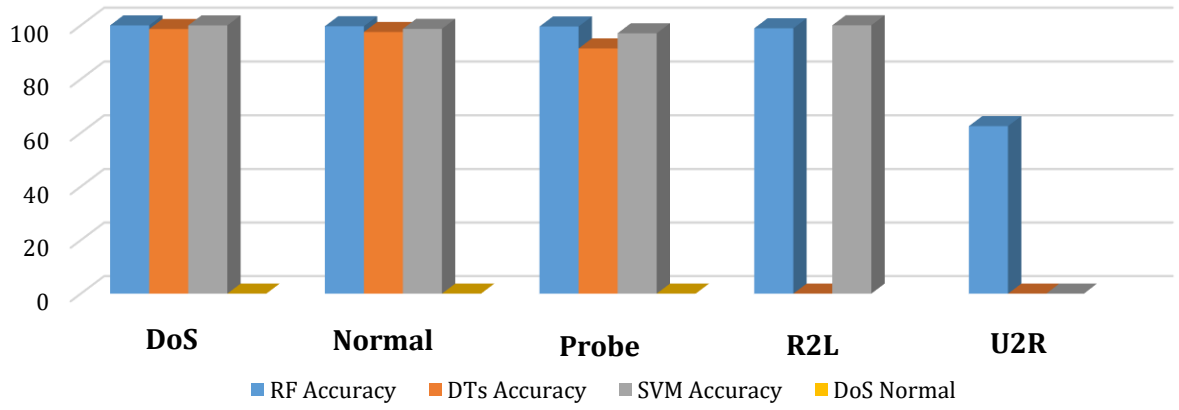


Figure 42 RF, DTs and SVM based on accuracy rate

Figure 43 shows the overall results for each class. The performance of the three base classifiers (RF, SVM and DTs) has brought considerable improvement on each class (DoS, normal, probe, R2L and U2R) and by combining class labels and appropriate features of each technique for the final result. RF has improved the precision and accuracy on U2R class (62.5%) and resolved the problem of data imbalance encountered by Kang and Cho (2006) and Zainal et al. 2009. Henceforth, ensemble classifier enables the NIDS to analyse the features of each class and to provide details of incoming packets at a very high rate and to reduce the possibility of future attacks.

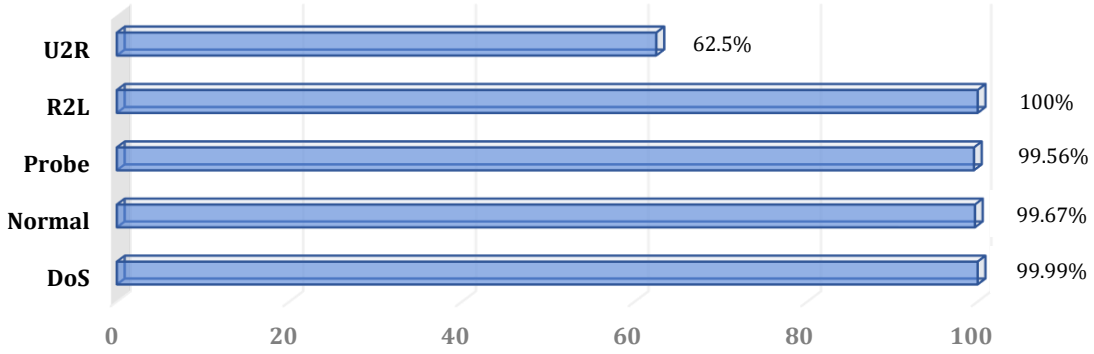


Figure 43 Ensemble classifier-performance

5.4 General discussion

The aim of the assessments was to find the best ML technique on several attributes of the five classes. The assessments of Boruta on the KDD99 were not quite satisfying due to time constraints. Overall, tables 29 and 33 show the performances of RF outperforming DTs and SVM. Both DTs and SVM are almost at the same level. In general, compared to previous works on the field of NIDS, the performances of DTs and SVM on R2R are not satisfying. On one hand, this may be due to the functionality specific to the U2R class, which may not be well selected. On the other hand, this may also be due to the problem of data imbalance. According to Kang and Cho (2006), data imbalance is one of the causes that degrade the performance of ML techniques in classifications (Kang and Cho, 2006). Data imbalance occurs when the number of variables in a class is much larger or smaller than that of other classes (Zainal et al., 2009). However, this work shows that both SVM and DTs can still perform well and overcome the problem of data imbalance encountered by Kang and Cho (2006), Zainal et al. 2009 and Balon-Perin (2012). Figure 41 shows the accuracy of the classifiers enhancing the average on each class. 92.34% on the DoS, 90.43% on the normal, 87.35% on the probe, 81.25% of R2L and 62.5% on the U2R. Figure 42 shows the accuracy rate of the three classifiers compared on each class. In general, RF, DTs, SVM performed well on U2R compared to previous works.

Figure 43 shows the final performance with 100% accuracy on R2L and 62% accuracy on U2R. 99.56% accuracy on the probe, 99.67% on the normal and 99.99 on the DoS classes. Particularly, malicious security threats such as DDoS based on volume flooding, intrusion flows and packets will be highly will be highly detected (99.99%). In general, the final performance seems to be acceptable for all the 5 classes. Although some of the attributes such as land and ipsweep were undetected, the final ensemble (Table 35) has significantly improved the overall results and also minimized the number of FPs and FNs. It can therefore be concluded that implementing RF, DTs and SVM classifiers for the NIDP is an effective proposition. SVM and DTs performed quite well. However, RF was able to improve the detection rate of each class and also improved the accuracy of the final test. Henceforth, it is clear that such fast and efficient classifiers can be implemented and integrated into the NIDP. After implementing these classifiers, it is now possible to answer the research question outlined in Chapter 1 (Section 1.2.2): ensemble classifier is a viable technique for addressing the NIDP, improving the accuracy and overall results. In terms of speed and ROC-AUC, RF outperformed DTs and SVM. In terms of feature selection, implementations using PCA performed better than Boruta. Overall, both ensemble classifier and feature selection algorithms can be systematically applied into the NIDP. Note that performance may vary depending on the dynamic architecture of NIDS.

5.5 Summary

The purpose of this chapter was to analyse and discuss the overall results. In order to achieve this, it was necessary to examine the reliability of the NIDS by comparing the tree classifiers in terms of prediction and accuracy. This chapter presented an exploratory analysis of various attributes to identify clear patterns in overall results and the classifier that outperformed others. The results reveal that RF outperformed DTs and SVM with 99% of accuracy. The next chapter (chapter 6), concludes this work, outlines various challenges in NIDS and presents the future works for the NIDS.

Chapter 6-Conclusion and Future works

If a man will begin with certainties, he will end in doubts; but if he will be content to begin with doubts, he will end in certainties…

-Francis Bacon-

6.1 Conclusion

The aim of this dissertation was to apply ensemble techniques into the NIDP by identifying normal data from abnormal ones, reducing the number of FNs and FPs, enhancing the system network protocols via test accuracy on various classes and implementing efficient algorithms to address the problem of intrusion detection. The use of ensemble learning has been introduced, implemented and analysed in this work. The approach aimed at developing a NIDS based-ensemble technique that would be more accurate and low in false alarms (for probe, U2R, R2L, normal and DoS classes). The attributes selected from the input packets were established and defined. A classifier was used to compute the variables of each class of attack. Acceptance of attributes for each class as specified by the control variables was established by the data processor. In order to create an ensemble of precise and varied classifiers, the methodology was based on training tree classifiers using the same training sets, but with several subsets of features. These classifiers were executed several times, each time with a different subset of the training data.

Particularly, this work showed that various features and subsets identified by Mukkamala et al. (2005) are different for each class of attacks. This work and the works conducted by Balon-Perin (2012) and Zainal et al. (2009) differ in terms of approach, techniques and evaluation of the proposed methodology. Balon-Perin (2012) applied minimum Redundancy Maximum Relevance (mRMR) and PCA algorithms while Zainal et al. (2009) used Rough Set Technique (RST) and Binary Particle Swarm Optimization algorithms (BPSO). However, they have divided both attacks into five distinct sub-problems and each problem was examined by a specific classifier. In general, some of the ML techniques employed by Zainal et al. (2009) and Balon-Perin (2012) failed to perform well; their performances were poor for DoS and U2R classes. On one hand, this may be due to the weights assign to each ML technique and the ensemble combination strategies. One the other hand, this may also be due to the integration of different data formats within the experimentations.

In terms of methodology, the model developed in this work has enhanced the attributes identified by Mukkamala et al. (2005). A sufficient number of features were available within the training and testing sets, which was substantial and has allowed the experiments to be performed on the complete set. The number of attributes selected in each class of attack was reversibly equivalent to the percentage of attributes in the original KDD99 dataset. Moreover, the NSL[5] (NSL-KDD) datasets enabled the classifiers to produce unbiased results by removing duplicate attributes. The subset selected by PCA gave good results, except for the U2R class for which the subset used by DTs and SVM was different from RF. The subset selected by Boruta gave interesting results in terms of compatibility of instances between the training sets and testing sets of the KDD99 dataset. In general, the two feature selection algorithms (PCA and Boruta) achieved good results in reduction of labelled training data and redundant attributes.

Furthermore, this work developed a combinatorial classifier based on combining the class labels of each algorithms (RF, DTs and SVM). It was then necessary to classify the incoming network traffic and to distinguish normal access to attack. Based on 4,898,431 records and 41 variables illustrated by both KDD99 and NSL datasets, 99% of accuracy was obtained as detection precision by fusing the RF, DTs and SVM. The results are quite satisfactorily comparing to the work conducted by Kevric et al. (2017) where 89.24% of accuracy was obtained by combining the random tree and NBTree algorithms based on the sum rule strategy; the results showed the highest performance obtained by using the complete NSL datasets. Typically, comparing to the approach introduced by Jabbar et al. (2016), this work corroborates that the combination strategies of final ensemble classifier outperforms the single classifier in terms of prediction, ROC-AUC and accuracy.

Finally, this work has demonstrated that the ensemble approach can enhance the prediction, the separability measures and the detection accuracy of a single classifier by assigning weights to each classifier in the final model, testing the NIDS and improving the network protocols. The robustness and effectiveness of the classifiers have been verified on the standard KDD99 and NSL KDD datasets in term of performance for the NIDS. Based on the results, DTs and SVM have performed well in all the classes and provided details of incoming packets (R2R, probe and normal) at a very high rate. However, RF shows a higher performance (ROC-AUC and accuracy rate) in U2R and DoS. Thus, by incorporating the RF in the final model, the performance has improved considerably.

[5] Is a new version of the KDD99 dataset (Dhanabal and Shantharajah ,2015)

6.1.1 Challenges in NIDS

The NIDS is complex and broad. Some of the problems and challenges in NIDS are listed below:

- NIDS requires significant expertise, both organizational and technical (basic knowledge of the specific network environment) (Werlinger et al., 2008).
- NIDS requires a huge amount of time and work during the installation and monitoring phases (Werlinger et al., 2008).
- Requesting resources when configuring the NIDS pre-processing and configuration steps (Werlinger et al., 2008).
- NIDS requires minimizing the number of FP and FN and maximizing accuracy (Anuar et al., 2008).
- Insufficient data set and integration of different data formats (attack instances should be tested in updated data sets) (Beigh et al., 2013).
- Some NIDS platforms are free source and commercial and present poor design (Beigh et al., 2013).
- Some detection techniques of the NIDS are incompetent (techniques should be precise enough to demarcate unknown attacks from known attacks and useful information) (Beigh et al., 2013).

6.2 Future work

This work aimed at developing a NIDS based-ensemble detection engine on the rules generated by each classifier. Besides, this dissertation may be extended by improving certain aspects of the approach and methodology. Some suggestions for a more promising of further research are listed below:

- The ML techniques introduced in this work can also be compared to other ensemble approaches such as ANN, Bayesian, K-NN, and NBTree.
- Another possibility for future research is to identify new features that can further improve the NIDS performance and minimize the number of packet losses.
- Using ensemble learning for the network-based intrusion prevention systems (NIPSs). NIPSs may be able to prevent a wider range of attacks and
- Identifying normal and attack instances by using new algorithms and more datasets such as the University of New Mexico (UNM) dataset, Australian Defence Force Academy-Linux Dataset (ADFA-LD) and Windows Dataset (WD), etc.

References

Anderson J. (1995). An introduction to neural networks, MIT Press, Cambridge, Bradford Book.

Anderson, J. P. (1980). Computer security threat monitoring and surveillance. Technical Report, James P. Anderson Company.

Albag, H. (2001). Network & agent based intrusion detection systems. TU Munich Dep. of Computer Science, Istanbul Tecical. University.

Anuar, N. B., Sallehudin, H., Gani, A., & Zakaria, O. (2008). Identifying false alarm for network intrusion detection system using hybrid data mining and decision tree. Malaysian journal of computer science, 21(2), 101-115.

Andrej, K., Janez, B., & Andrej, K. (2011). Introduction to the Artificial Neural Networks. Artificial Neural Networks-Methodological Advances and Biomedical Applications.

Ahmad, I., Basheri, M., Iqbal, M. J., & Raheem, A. (2018). Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection. IEEE Access.

Aizerman, M. A. (1964). Theoretical foundations of the potential function method in pattern recognition learning. Automation and remote control, 25, 821-837.

Achirul Nanda, M., Boro Seminar, K., Nandika, D., & Maddu, A. (2018). A comparison study of kernel functions in the support vector machine and its application for termite detection. Information, 9(1), 5.

Al-Garadi, M. A., Mohamed, A., Al-Ali, A., Du, X., & Guizani, M. (2018). A survey of machine and deep learning methods for internet of things (IoT) security. arXiv preprint arXiv:1807.11023.

Bay, S. D., Kibler, D., Pazzani, M. J., & Smyth, P. (2000). The UCI KDD archive of large data sets for data mining research and experimentation. ACM SIGKDD explorations newsletter, 2(2), 81-85 (http://kdd.ics.uci.edu).

Balon-Perin, A. (2012). Ensemble-based methods for intrusion detection (Master's thesis, Institutt for datateknikk og informasjonsvitenskap).

Balon-Perin, A., & Gambäck, B. (2013). Ensembles of decision trees for network intrusion detection systems. International Journal on Advances in Security, 6(1 & 2).

Borji, A. (2007, December). Combining heterogeneous classifiers for network intrusion detection. In Annual Asian Computing Science Conference (pp. 254-260). Springer, Berlin, Heidelberg.

Breiman, L. (1996). Bagging predictors. Machine learning, 24(2), 123-140.

Breiman, L. (2001). Random forests. Machine learning, 45(1), 5-32.

Breiman, L. (2003, May). RF/tools: A class of two-eyed algorithms. In SIAM workshop (pp. 1-56).

Bouzida, Y., Cuppens, F., Cuppens-Boulahia, N., & Gombault, S. (2004, June). Efficient intrusion detection using principal component analysis. In 3éme Conférence sur la Sécurité et Architectures Réseaux (SAR), La Londe, France (pp. 381-395).

Bouzida, Y., & Cuppens, F. (2006, September). Neural networks vs. decision trees for intrusion detection. In IEEE/IST Workshop on Monitoring, Attack Detection and Mitigation (MonAM) (pp. 81 88).

Beigh, B. M., Bashir, U., & Chahcoo, M. (2013). Intrusion Detection and Prevention System: Issues and Challenges. International Journal of Computer Applications, 76(17).

Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. Data mining and knowledge discovery, 2(2), 121-167.

Cowan, C., Wagle, F., Pu, C., Beattie, S., & Walpole, J. (2000). Buffer overflows: Attacks and defenses for the vulnerability of the decade. In DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings(Vol. 2, pp. 119-129). IEEE.

Cortes, C., & Vapnik, V. (1995). Support-vector network-. Machine Learning 20: 273–297. Portfolio Selection, Journal of Global Optimization, 43(2-3).

Cover, T. M., & Van Campenhout, J. M. (1977). On the Possible Orderings in the Measurement Selection Problem. IEEE Trans. Systems, Man, and Cybernetics, 7(9), 657-661.

Chih-Fong Tsai, Yu-Feng Hsu, Chia-Ying Lin & Wei-YangLin (2009). Intrusion detection by machine learning: A review. Expert Systems with Applications, Volume 36, Issue 10, pp. 11994-12000.

Chen, W. H., Hsu, S. H., & Shen, H. P. (2005). Application of SVM and ANN for intrusion detection. Computers & Operations Research, 32(10), 2617-2634.

Creech G. and J. Hu. (2013) Generation of a new IDS test dataset: Time to retire the KDD collection. In Wireless Communications and Networking Conference (WCNC), IEEE, pages 44874492, 2013.

Chapelle, O., Scholkopf, B., & Zien, A. (2009). Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. IEEE Transactions on Neural Networks, 20(3), 542-542.

Chollet, F., & Allaire, J. J. (2018). Deep Learning with R. Manning Publications Company.

Dietterich T.G. (2000) Ensemble Methods in Machine Learning. In: Multiple Classifier Systems. MCS 2000. Lecture Notes in Computer Science, vol 1857. Springer, Berlin, Heidelberg.

Dhillon, I., Mallela, S., & Kumar, R. (2002). Information theoretic feature clustering for text classification. In JOURNAL OF MACHINE LEARNING RESEARCH (JMLR), SPECIAL.

Dhanabal, L., & Shantharajah, S. P. (2015). A study on NSL-KDD dataset for intrusion detection system based on classification algorithms. International Journal of Advanced Research in Computer and Communication Engineering, 4(6), 446-452.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). Pattern classification. 2nd. Edition. New York, 55.

Denning, D. E. (1987). An intrusion-detection model. IEEE Transactions on software engineering, (2), 222-232.

Debar, H., Dacier, M., & Wespi, A. (1999). Towards a taxonomy of intrusion-detection systems. Computer Networks, 31(8), 805-822.

Debar, H., Dacier, M., & Wespi, A. (2000, July). A revised taxonomy for intrusion-detection systems. In Annales des télécommunications (Vol. 55, No. 7-8, pp. 361-378). Springer-Verlag.

Efron, B., & Tibshirani, R. (1993). An Introduction to the Bootstrap. Chapman and Hall, New York.

Fan, W., Lee, W., Stolfo, S. J., & Miller, M. (2000, May). A multiple model cost-sensitive approach for intrusion detection. In European conference on machine learning (pp. 142-154). Springer, Berlin, Heidelberg.

Fletcher, R. (2013). Practical methods of optimization. John Wiley & Sons.

Freund, Y., & Schapire, R. (1996). Experiments with a new boosting algorithm. In Proceedings of the Thirteenth International Conference on Machine Learning, pp. 148–156 Bari, Italy.

Graham, R. Mar. 21, (2000). "FAQ: Network Intrusion Detection Systems." RobertGraham.com Homepage. Robert Graham.

Haq, N. F., Onik, A. R., Hridoy, M. A. K., Rafni, M., Shah, F. M., & Farid, D. M. (2015). Application of machine learning approaches in intrusion detection system: a survey. IJARAI-International Journal of Advanced Research in Artificial Intelligence, 4(3), 9-18.

Hansen, L. K., & Salamon, P. (1990). Neural network ensembles. IEEE transactions on pattern analysis and machine intelligence, 12(10), 993-1001.

Heberlein, L. T., Dias, G. V., Levitt, K. N., Mukherjee, B., Wood, J., & Wolber, D. (1990, May). A network security monitor. In Research in Security and Privacy, 1990. Proceedings., 1990 IEEE Computer Society Symposium on(pp. 296-304). IEEE.

Haijun, X., Fang, P., Ling, W., & Hongwei, L. (2007, November). Ad hoc-based feature selection and support vector machine classifier for intrusion detection. In 2007 IEEE International Conference on Grey Systems and Intelligent Services (pp. 1117-1121). IEEE.

Jabbar, M. A., Srinivas, K., & Reddy, S. S. S. (2016, December). A Novel Intelligent Ensemble Classifier for Network Intrusion Detection System. In International Conference on Soft Computing and Pattern Recognition (pp. 490-497). Springer, Cham.

Jabbar, M. A., Aluvalu, R., & Reddy, S. (2017, February). Cluster based Ensemble classification for Intrusion Detection System. In Proceedings of the 9th International Conference on Machine Learning and Computing (pp. 253-257). ACM.

Jabbar, M. A., & Aluvalu, R. (2017). RFAODE: A Novel Ensemble Intrusion Detection System. Procedia Computer Science, 115, 226-234.

Jain, A., & Zongker, D. (1997). Feature selection: Evaluation, application, and small sample performance. IEEE transactions on pattern analysis and machine intelligence, 19(2), 153-158.

Jain, A. K., Duin, R. P. W., & Mao, J. (2000). Statistical pattern recognition: A review. IEEE Transactions on pattern analysis and machine intelligence, 22(1), 4-37.

Jang, J.-S., Sun, C.-T., & Mizutani, E. (1996). Neuro-fuzzy and soft computing: A computational approach to learning and machine intelligence. New Jersey: Prentice Hall.

Kang, P., & Cho, S. (2006, October). EUS SVMs: Ensemble of under-sampled SVMs for data imbalance problems. In International Conference on Neural Information Processing (pp. 837-846). Springer, Berlin, Heidelberg.

Kendall, K. K. R. (1999). A database of computer attacks for the evaluation of intrusion detection systems (Doctoral dissertation, Massachusetts Institute of Technology).

Kittler, J. Hatef, M., Duin, R. P. W., & Matas, J. (1998). On combining classifiers. IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(3), 226–239.

Kumar, P. A. R., & Selvakumar, S. (2013). Detection of distributed denial of service attacks using an ensemble of adaptive and hybrid neuro-fuzzy systems. Computer Communications, 36(3), 303-319.

Kumar, B. S., Ch, T., Raju, R. S. P., Ratnakar, M., Baba, S. D., & Sudhakar, N. (2013). Intrusion detection system-types and prevention.

Kuncheva, L. I., Bezdek, J. C., & Duin, R. P. (2001). Decision templates for multiple classifier fusion: an experimental comparison. Pattern recognition, 34(2), 299-314.

Kursa, M. B., & Rudnicki, W. R. (2010). Feature selection with the Boruta package. J Stat Softw, 36(11), 1-13.

Kevric, J., Jukic, S., & Subasi, A. (2017). An effective combining classifier approach using tree algorithms for network intrusion detection. Neural Computing and Applications, 28(1), 1051-1058.

Lee, W. and Stolfo S.J. (2000). A framework for constructing features and models for intrusion detection systems. ACM Trans. on Inform. And System Security, 3(4), 227-261.

Levin, S. L., & Schmidt, S. (2014). IPv4 to IPv6: Challenges, solutions, and lessons. Telecommunications Policy, 38(11), 1059-1068.

Lokesak. B (2008). "A Comparison Between Signature Based and Anomaly Based Intrusion Detection Systems"(PPT). www.iup.edu.

Li, W. (2004). Using genetic algorithm for network intrusion detection. Proceedings of the United States Department of Energy Cyber Security Group, 1, 1-8.

Liu, H., & Yu, L. (2005). Toward integrating feature selection algorithms for classification and clustering. IEEE Transactions on knowledge and data engineering, 17(4), 491-502.

Liu, G., Yi, Z., & Yang, S. (2007). A hierarchical intrusion detection model based on the PCA neural networks. Neurocomputing, 70(7-9), 1561-1568.

Liaw, A., & Wiener, M. (2002). Classification and regression by randomForest. R News, 2 (3): 18–22. R package version 4.6. 10.

Laskov, P., Düssel, P., Schäfer, C., & Rieck, K. (2005, September). Learning intrusion detection: supervised or unsupervised?. In International Conference on Image Analysis and Processing (pp. 50-57). Springer, Berlin, Heidelberg.

Maiwald, E. (2001). Network security: a beginner's guide. McGraw-Hill Professional.

Mansour, Y. (1997, July). Pessimistic decision tree pruning based on tree size. In MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE- (pp. 195-201). MORGAN KAUFMANN PUBLISHERS, INC.

Mitchell, T. M. (1997). Machine learning (mcgraw-hill international editions computer science series).

Michalski, R. S., Bratko, I., & Kubat, M. (1998). Machine learning and data mining methods and applications. Chichester, New York, Weinheim, Brisbane, Toronto, Singapore: Wiley.

McHugh J., Christie A., Allen J., (2002). Defending Yourself: The Role of Intrusion Detection Systems. IEEE Software, Sept. /Oct. 2000, 42-51.

Mukkamala, S., Janoski, G., & Sung, A. (2002). Intrusion detection using neural networks and support vector machines. In Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on (Vol. 2, pp. 1702-1707). IEEE.

Mukkamala, S., & Sung, A. H. (2003). Identifying significant features for network forensic analysis using artificial intelligent techniques. International Journal of digital evidence, 1(4), 1-17.

Mukkamala, S., Sung, A., & Abraham, A. (2005). Cyber security challenges: Designing efficient intrusion detection systems and antivirus tools. Vemuri, V. Rao, Enhancing Computer Security with Smart Technology.(Auerbach, 2006), 125-163.

Oja, E. (1992). Principal components, minor components, and linear neural networks. Neural networks, 5(6), 927-935.

Papamartzivanos, D., Mármol, F. G., & Kambourakis, G. (2018). Dendron: Genetic trees driven rule induction for network intrusion detection systems. Future Generation Computer Systems, 79, 558-574.

Prathibha, K. S., Kumar, P., & Shyni, T. S. (2014). Analysis of hybrid intrusion detection system based on data mining techniques. Int J Eng Trends Technol (IJETT), 15(9), 447-452.

Pharate, A., Bhat, H., Shilimkar, V., & Mhetre, N. (2015). Classification of Intrusion Detection System. International Journal of Computer Applications, 118(7).

Pearson, K. (1901). LIII. On lines and planes of closest fit to systems of points in space. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 2(11), 559-572.

Perdisci, R., Ariu, D., Fogla, P., Giacinto, G., & Lee, W. (2009). McPAD: A multiple classifier system for accurate payload-based anomaly detection. Computer networks, 53(6), 864-881.

Polikar R. (2012) Ensemble Learning. In: Zhang C., Ma Y. (eds) Ensemble Machine Learning. Springer, Boston, MA

Porres, I., & Fernandez, M. D. M. (2008). An Evaluation of current IDS (Doctoral dissertation, M. Sc Thesis, Department of Electrical Engineering, at Linkoping Institute of Technology, Sweden).

Quinlan, J. R. (1987). Simplifying decision trees. International journal of man-machine studies, 27(3), 221-234.

Rokach, L. (2010). Ensemble-based classifiers. Artificial Intelligence Review, 33(1-2), 1-39.

Robnik-Šikonja, M., & Kononenko, I. (2003). Theoretical and empirical analysis of ReliefF and RReliefF. Machine learning, 53(1-2), 23-69.

Shon, T. and Moon, J. (2007) A hybrid machine learning approach to network anomaly detection, Information Sciences, 177, pp. 3799-3821.

Shoemaker L. and Hall L.O. (2011) Anomaly Detection Using Ensembles. In: Sansone C., Kittler J., Roli F. (eds) Multiple Classifier Systems. MCS 2011. Lecture Notes in Computer Science, vol 6713. Springer, Berlin, Heidelberg.

Sangkatsanee, P., Wattanapongsakorn, N., & Charnsripinyo, C. (2009). Real-time intrusion detection and classification. IEEE network, 1-5.

Stewart, J. M., Tittel, E., & Chapple, M. (2008). CISSP: Certified information systems security professional study guide. John Wiley & Sons.

Sommer, R., & Paxson, V. (2010, May). Outside the closed world: On using machine learning for network intrusion detection. In Security and Privacy (SP), 2010 IEEE Symposium on (pp. 305-316). IEEE.

Sweeney, M., Baumrucker, C. T., Burton, J. D., & Dubrawsky, I. (2003). Cisco security professional's guide to secure intrusion detection systems. Syngress Publishing.

Stoppiglia, H., Dreyfus, G., Dubois, R., & Oussar, Y. (2003). Ranking a random feature for variable and feature selection. Journal of machine learning research, 3(Mar), 1399-1414.

Song, L., Smola, A., Gretton, A., Borgwardt, K. M., & Bedo, J. (2007, June). Supervised feature selection via dependence estimation. In Proceedings of the 24th international conference on Machine learning (pp. 823-830). ACM.

Shanmugam, B., & Idris, N. B. (2011). Hybrid intrusion detection systems (HIDS) using Fuzzy logic. In Intrusion Detection Systems. InTech.

Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. Cambridge university press.

Shah, B., & Trivedi, B. H. (2012). Artificial neural network based intrusion detection system: A survey. International Journal of Computer Applications, 39(6), 13-18.

Sutton, R. S., & Barto, A. G. (2011). Reinforcement learning: An introduction.

Staudemeyer, R. C., & Omlin, C. W. (2014). Extracting salient features for network intrusion detection using machine learning methods. South African computer journal, 52(1), 82-96.

Szepesvári, C. (2010). Algorithms for reinforcement learning. Synthesis lectures on artificial intelligence and machine learning, 4(1), 1-103.

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009, July). A detailed analysis of the KDD CUP 99 data set. In Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on (pp. 1-6). IEEE.

Thanasekaran, R.D. (2011). A Robust and Efficient Real Time Network Intrusion Detection System Using Artificial Neural Network In Data Mining. International Journal of Information Technology Convergence and Services (IJITCS) Vol, 1.

Tang, J., Alelyani, S., & Liu, H. (2014). Feature selection for classification: A review. Data classification: Algorithms and applications, 37.

Triguero, I., García, S., & Herrera, F. (2015). Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. Knowledge and Information Systems, 42(2), 245-284.

Werlinger, R., Hawkey, K., Muldner, K., Jaferian, P., & Beznosov, K. (2008, July). The challenges of using an intrusion detection system: is it worth the effort?. In Proceedings of the 4th symposium on Usable privacy and security (pp. 107-118). ACM.

Wei, Z., Yu-xin, Z., Hao-yu, W., Xu, Z., & Ai-guo, W. (2010, August). Intrusive detection systems design based on BP neural network. In 2010 Ninth International Symposium on Distributed Computing and Applications to Business, Engineering and Science (pp. 462-465). IEEE.

Witten, I.H., Frank, E. & Hall, M.A. (2011). Data mining. 1st ed. Burlington, MA. USA: Elsevier.

Yu, J., Tao, D., & Lin, Z. (2016, August). A hybrid web log based intrusion detection model. In 2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS) (pp. 356-360). IEEE.

Zhou, Z. H. (2009). Ensemble. In L. Liu & T. Özsu (Eds.), Encyclopedia of database systems. Berlin: Springer.

Zainal, A., Maarof, M. A., & Shamsuddin, S. M. (2009). Ensemble classifiers for network intrusion detection system. Journal of Information Assurance and Security, 4(3), 217-225.

Appendices

Appendix A: The 41 features provided by the KDD99 dataset

Nr	Features	
	Name	Description
1	duration	duration of connection in seconds
2	protocol_type	connection protocol (tcp, udp, icmp)
3	service	dst port mapped to service (e.g. http, ftp, ..)
4	flag	normal or error status flag of connection
5	src_bytes	number of data bytes from src to dst
6	dst_bytes	bytes from dst to src
7	land	1 if connection is from/to the same host/port; else 0
8	wrong_fragment	number of 'wrong' fragments (values 0,1,3)
9	urgent	number of urgent packets
10	hot	number of 'hot' indicators (bro-ids feature)
11	num_failed_logins	number of failed login attempts
12	logged_in	1 if successfully logged in; else 0
13	num_compromised	number of 'compromised' conditions
14	root_shell	1 if root shell is obtained; else 0
15	su_attempted	1 if 'su root' command attempted; else 0
16	num_root	number of 'root' accesses
17	num_file_creations	number of file creation operations
18	num_shells	number of shell prompts
19	num_access_files	number of operations on access control files
20	num_outbound_cmds	number of outbound commands in an ftp session
21	is_hot_login	1 if login belongs to 'hot' list (e.g. root, adm); else 0
22	is_guest_login	1 if login is 'guest' login (e.g. guest, anonymous); else 0
23	count	number of connections to same host as current connection in past two seconds
24	srv_count	number of connections to same service as current connection in past two seconds
25	serror_rate	% of connections that have 'SYN' errors
26	srv_serror_rate	% of connections that have 'SYN' errors
27	rerror_rate	% of connections that have 'REJ' errors
28	srv_rerror_rate	% of connections that have 'REJ' errors
29	same_srv_rate	% of connections to the same service
30	diff_srv_rate	% of connections to different services
31	srv_diff_host_rate	% of connections to different hosts
32	dst_host_count	count of connections having same dst host
33	dst_host_srv_count	count of connections having same dst host and using same service
34	dst_host_same_srv_rate	% of connections having same dst port and using same service
35	dst_host_diff_srv_rate	% of different services on current host
36	dst_host_same_src_port_rate	% of connections to current host having same src port
37	dst_host_srv_diff_host_rate	% of connections to same service coming from diff. hosts
38	dst_host_serror_rate	% of connections to current host that have an S0 error
39	dst_host_srv_serror_rate	% of connections to current host and specified service that have an S0 error
40	dst_host_rerror_rate	% of connections to current host that have an RST error
41	dst_host_srv_rerror_rate	% of connections to the current host and specified service that have an RST error

(Staudemeyer and Omlin , 2014).

Appendix B: Code (Excerpt_RStudio)

#RF,DTs,SVM

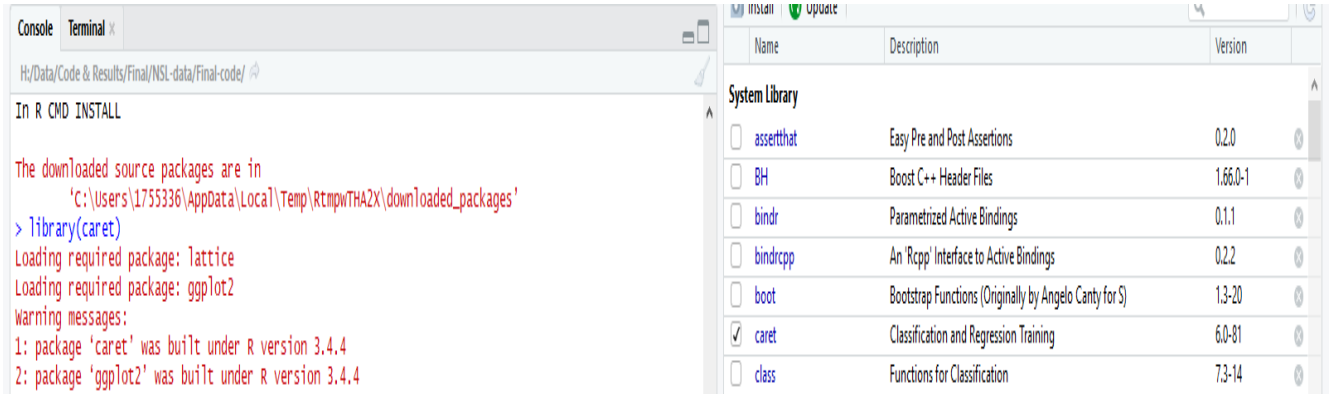
#*http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html* #(Staudemeyer and Omlin, 2014).
---for the KDD99-Datasets-description.

#*http://www.unb.ca/cic/datasets/nsl.html* #(Dhanabal and Shantharajah ,2015).

---for the NSL-KDD_Datasets-description.

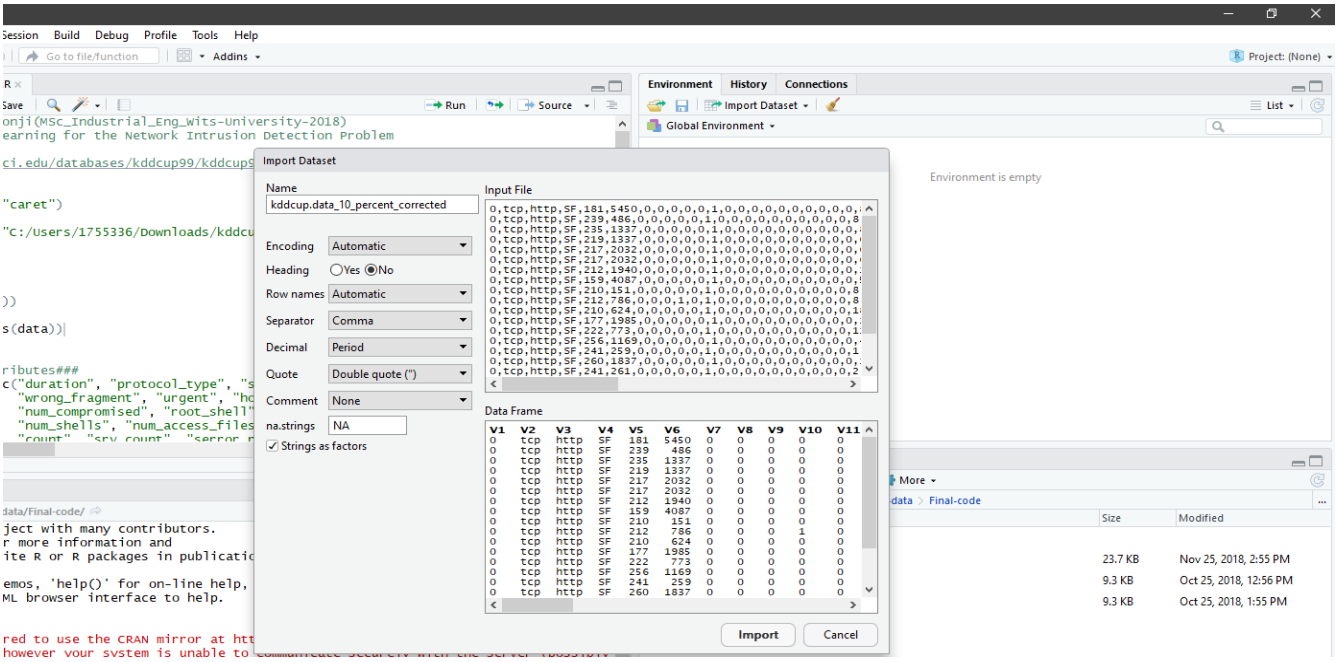
install.packages("caret") (classification and regression training)

library(caret)

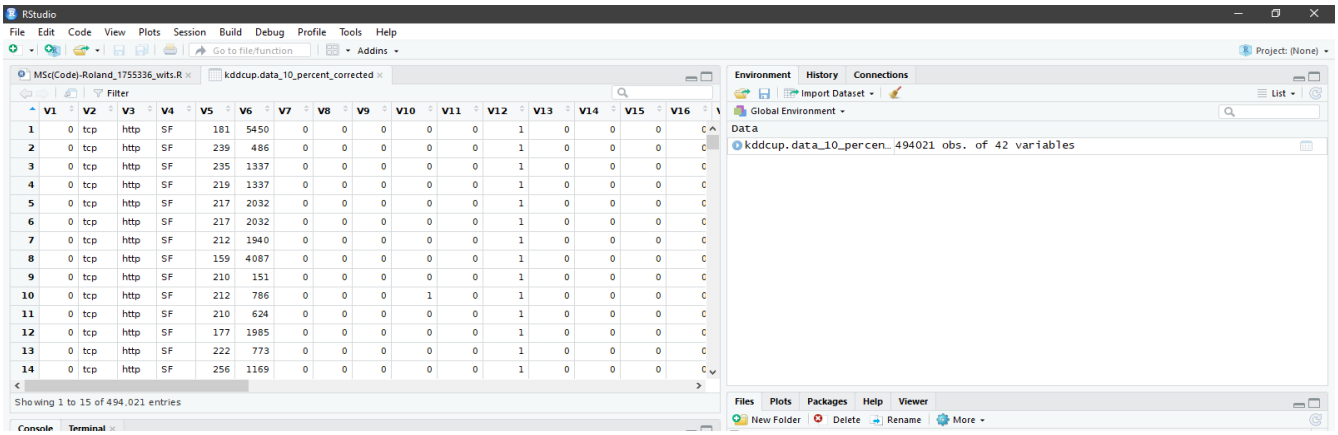


(1)KDD-_datasets

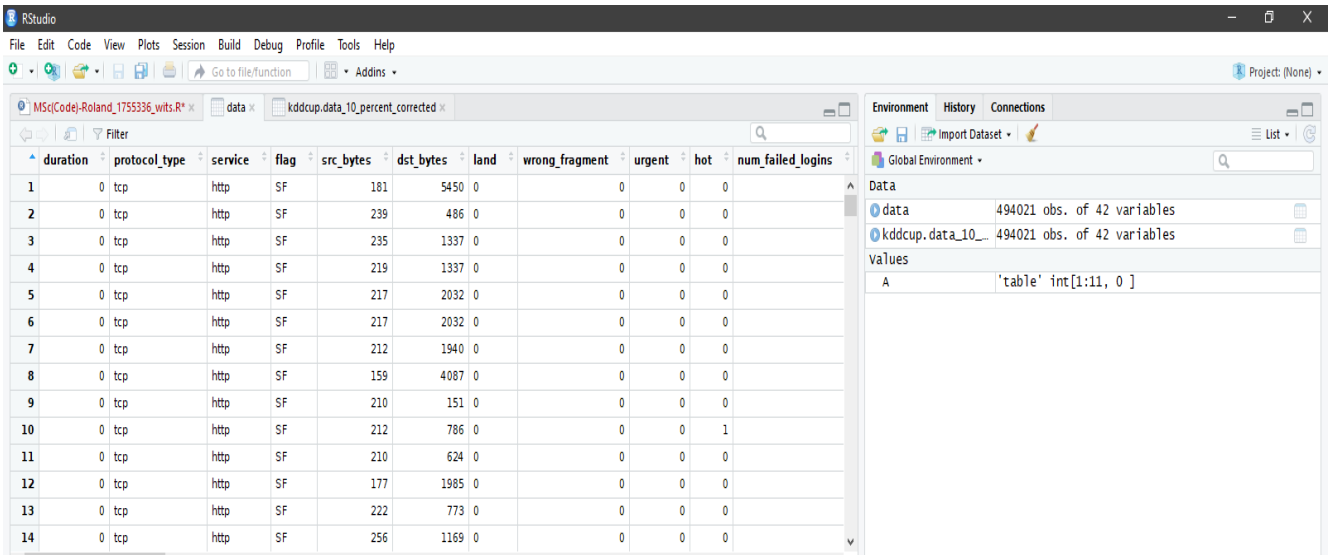
data<-read.table("C:/Users/1755336/Downloads/kddcup.data_10_percent_corrected.csv", sep =",")



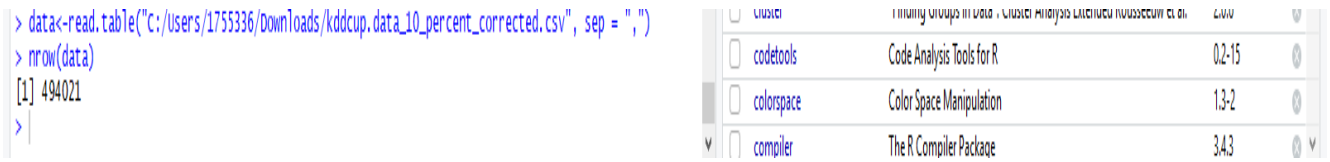
kddcup_data#



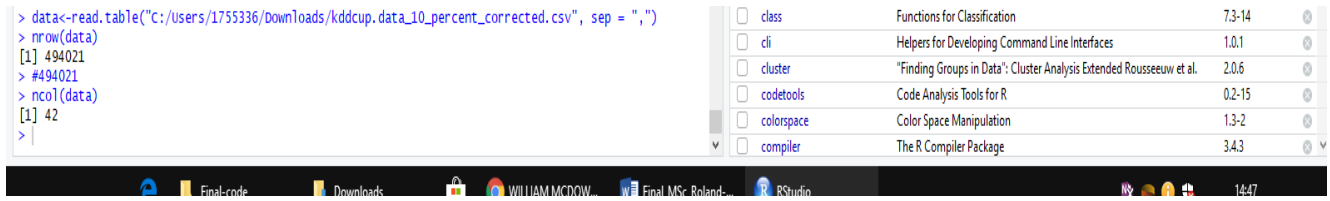
View(data)



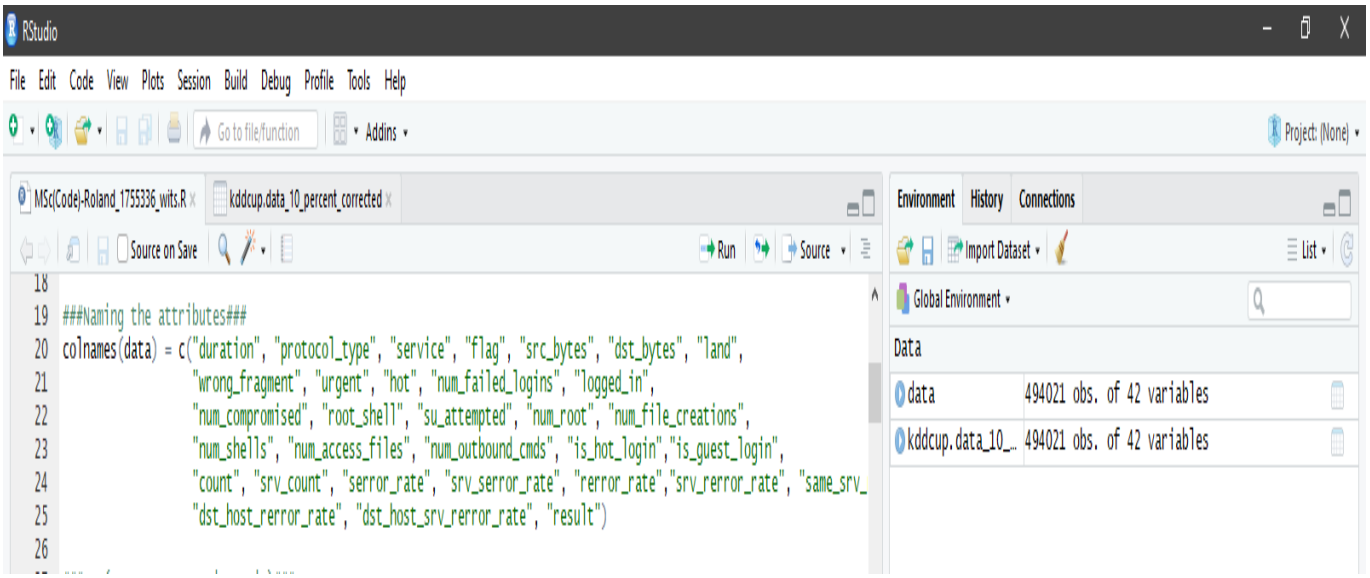
nrow(data)



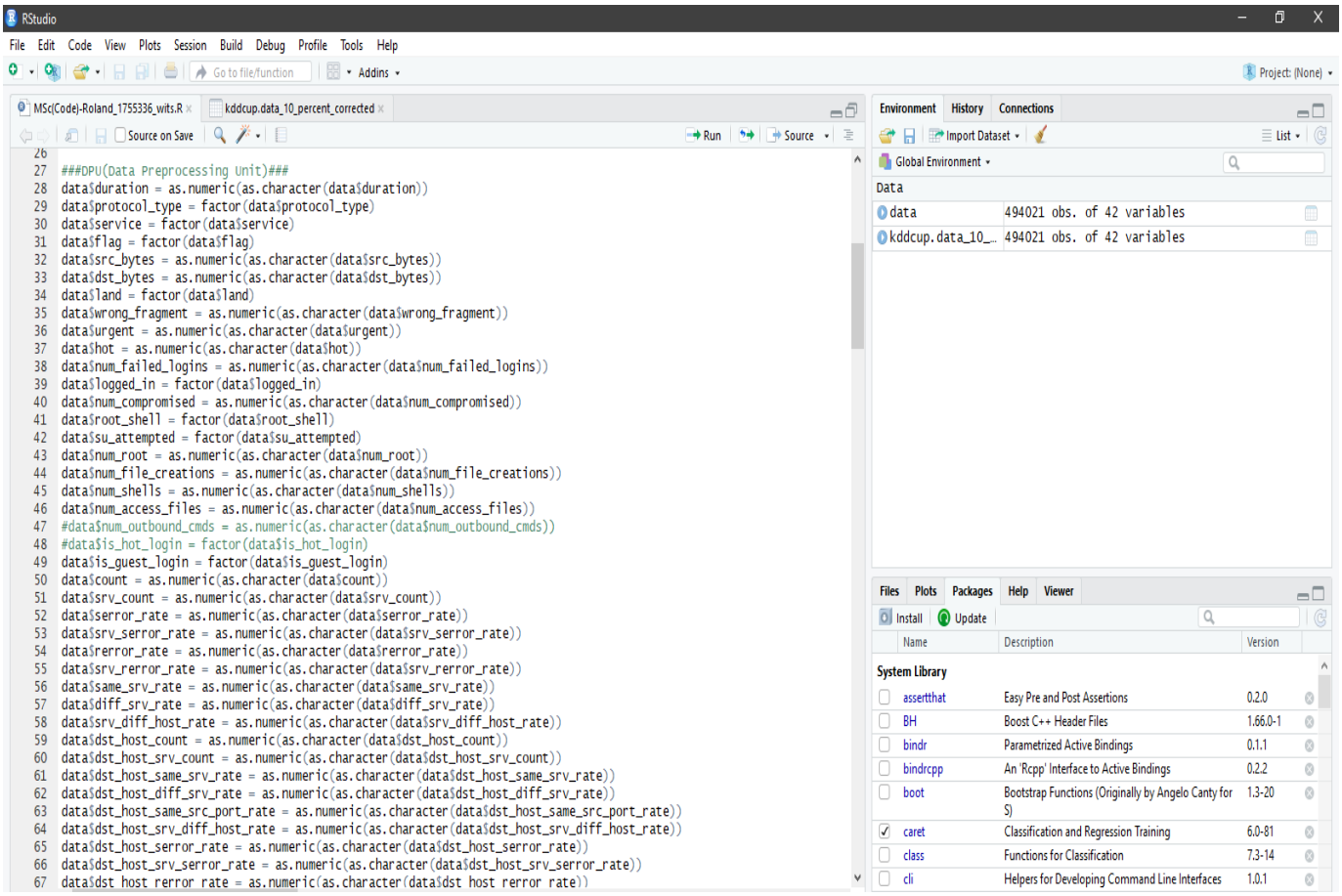
ncol(data)



###Naming the attributes###



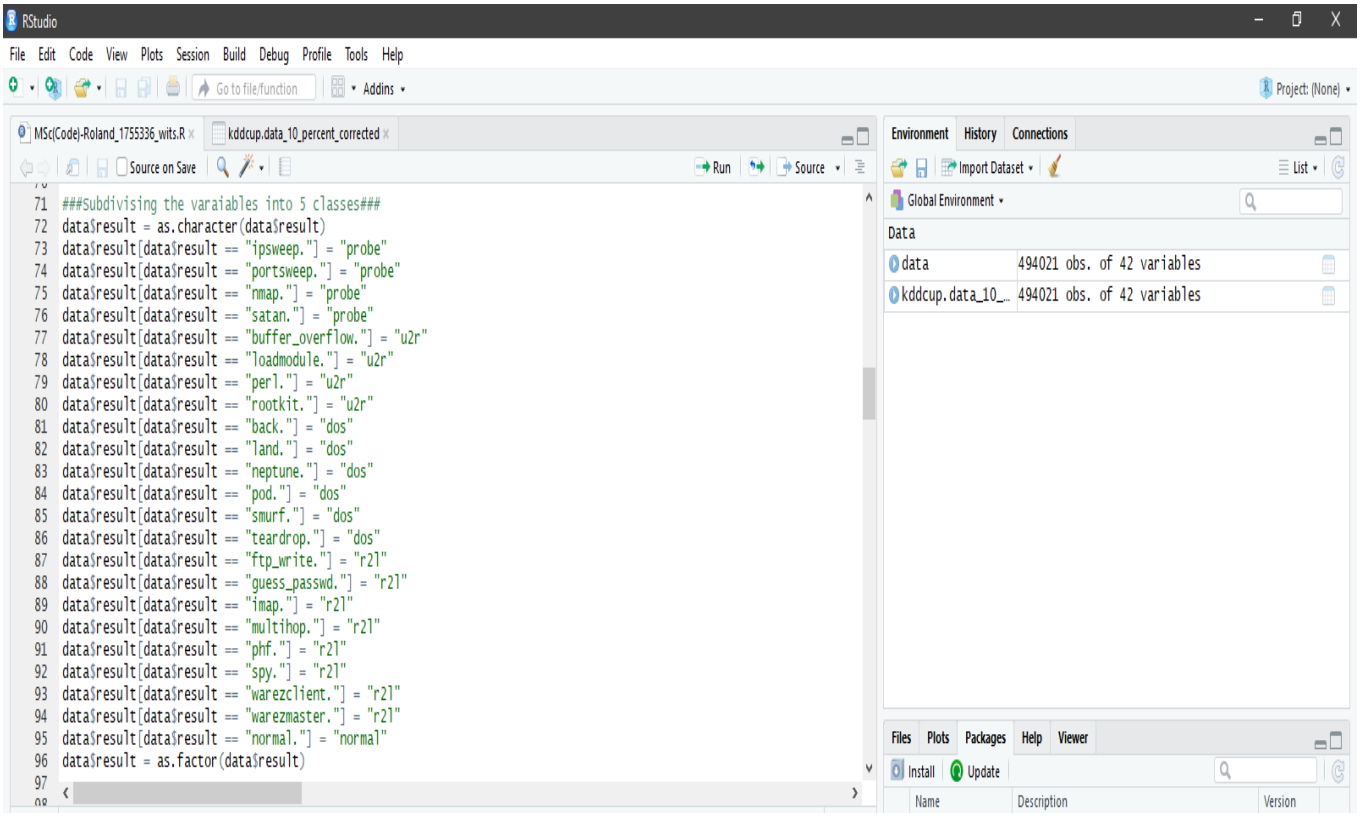
###DPU###



#DPU-Results#

###Subdivising the varaiables into 5 classes###

#DoS#Probe#U2R#U2L#Normal#

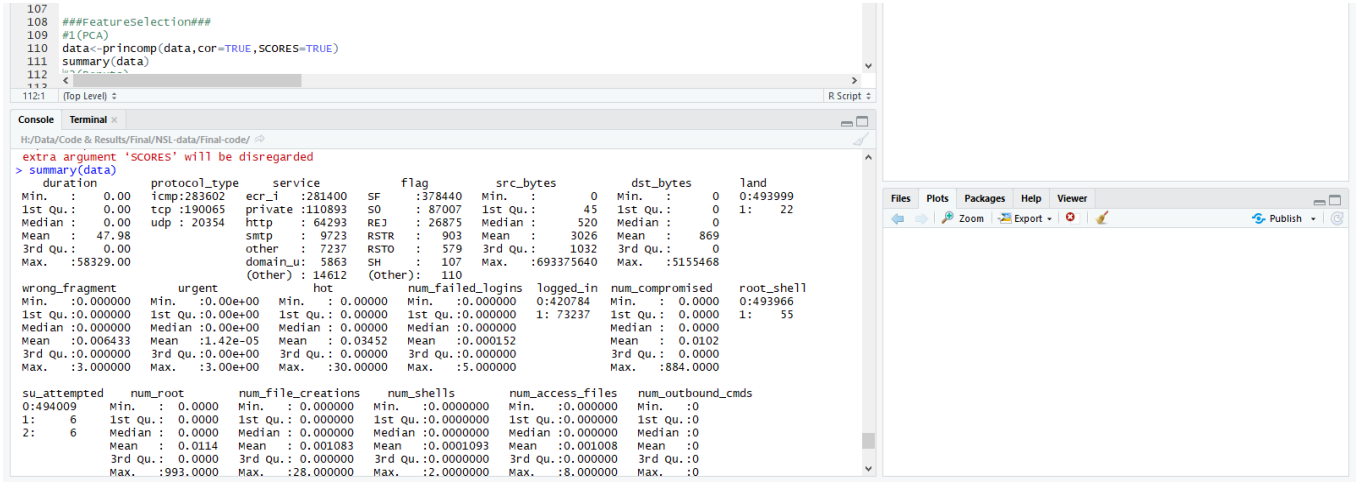


###FeatureSelection###

#1(PCA)

data<-princomp(data,cor=TRUE,SCORES=TRUE)

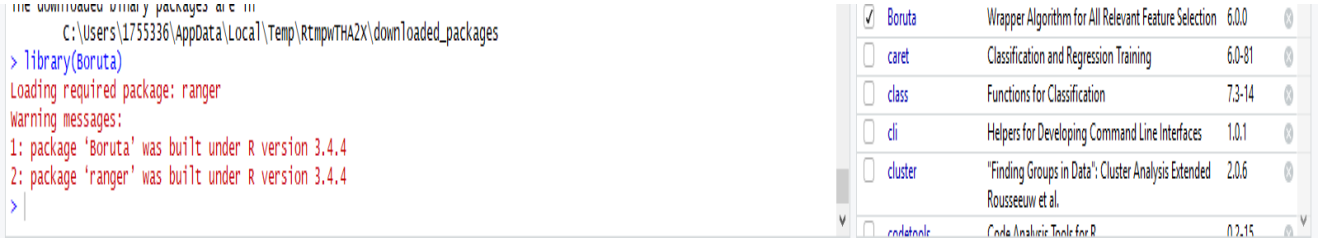
summary(data)



#2(Boruta)

install.packages("Boruta")

library(Boruta)



boruta.train <- Boruta(result~.-service, data = data,doTrace = 2,maxRuns=25)

print(boruta.train)

+1.2 hours boruta performed 12 iterations on the KDD99:

- 36 attributes confirmed important: count, diff_srv_rate, dst_bytes, dst_host_count, dst_host_diff_srv_rate and 31 more.
- 2 attributes rejected: is_hot_login, num_outbound_cmds; still have 2 attributes left.

2.30092 hours, boruta performed 24 iterations:

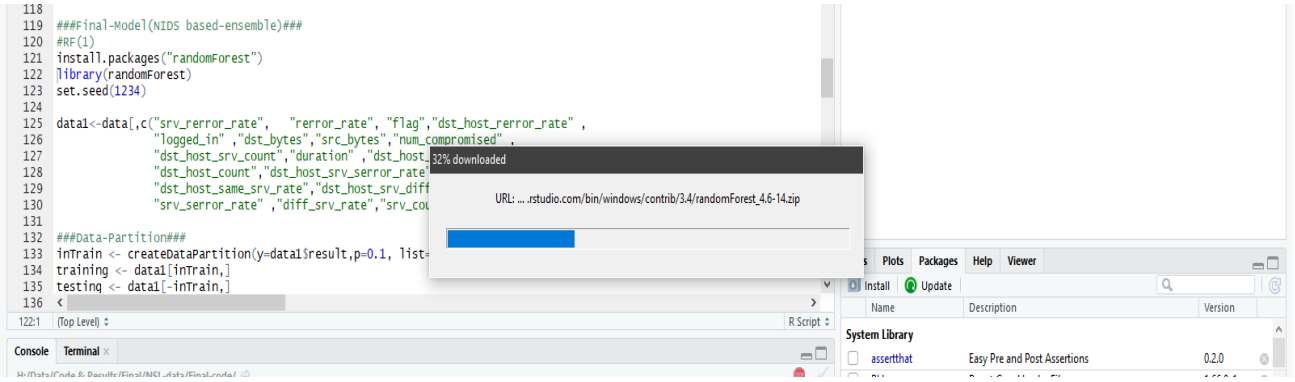
- 36 attributes confirmed important: count, diff_srv_rate, dst_bytes, dst_host_count, dst_host_diff_srv_rate and 31 more.
- 3 attributes confirmed unimportant: is_hot_login, num_outbound_cmds, urgent.
- 1 tentative attributes left: su_attempted.

###NIDS based-ensemble)###

#RF(1)

install.packages("randomForest")

library(randomForest)



set.seed(1234)

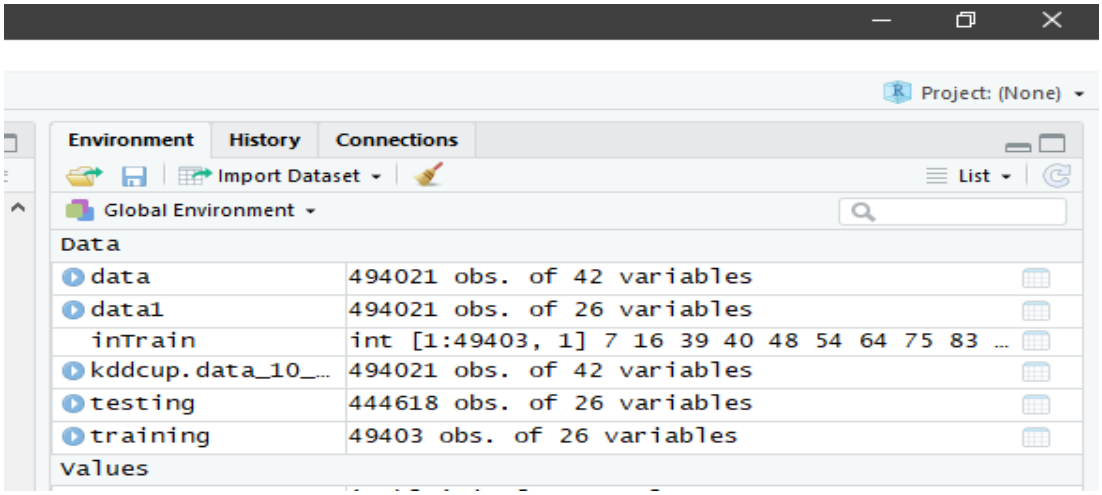
###Data-Partition###

inTrain <- createDataPartition(y=data1$result,p=0.1, list=FALSE)

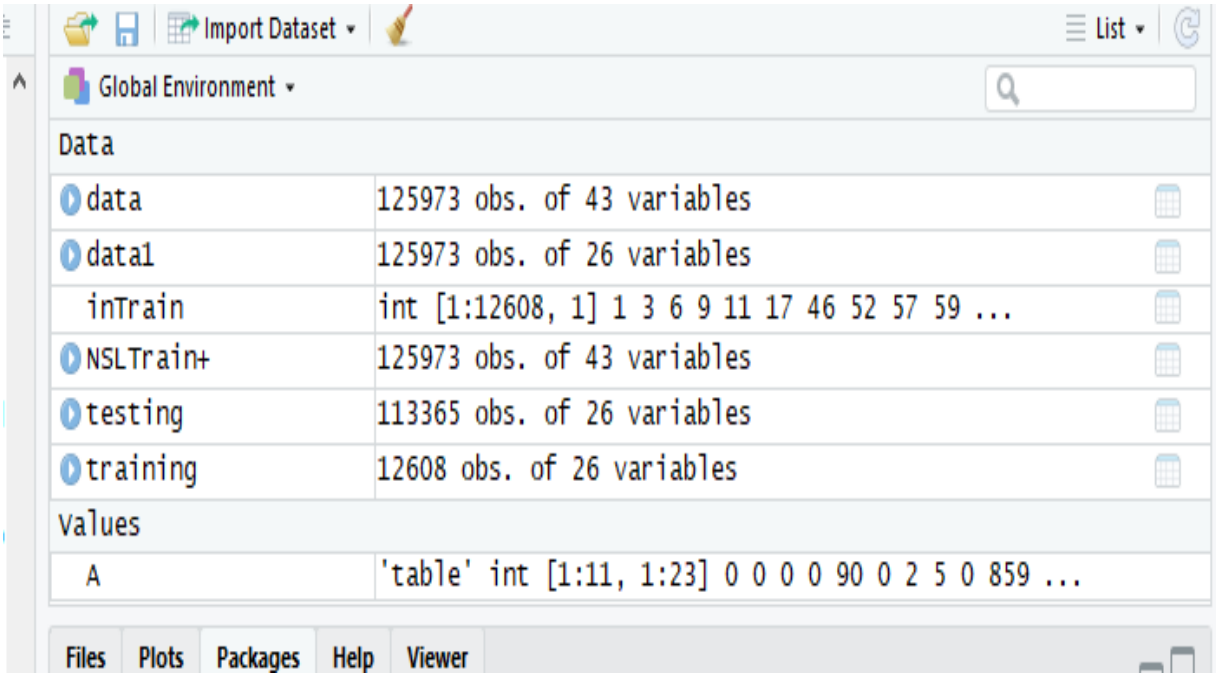
training <- data1[inTrain,]

testing <- data1[-inTrain,]

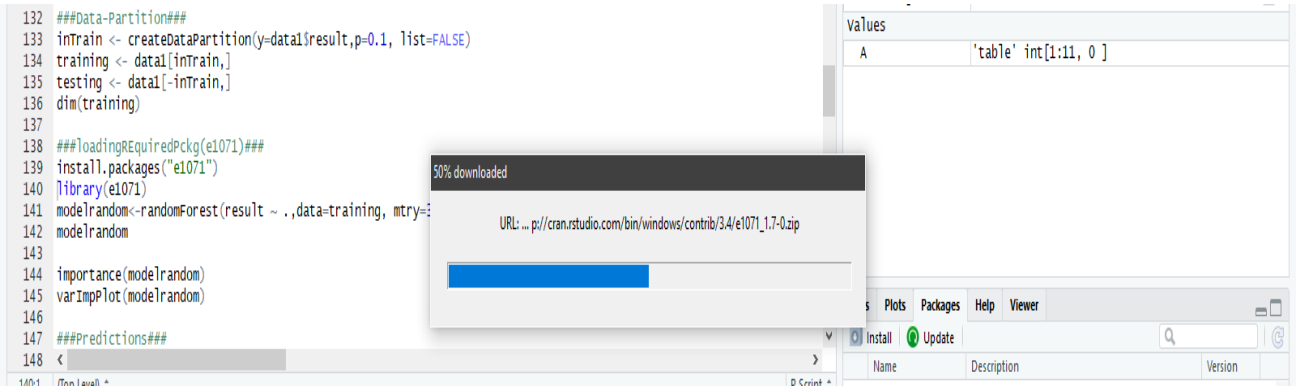
dim(training)



#For-NSL-KDD#



###loadingREquiredPckg(e1071)###



install.packages("e1071")

library(e1071)

modelrandom<-randomForest(result ~ .,data=training, mtry=3, ntree=20)

modelrandom

importance(modelrandom)

varImpPlot(modelrandom)

###Predictions###

pred <- predict(modelrandom,testing);

modelrandom

###ConfusionMatrix###

testing$predRight <- pred==testing$result

table(pred,testing$result)

A=table(pred,testing$result)

round(prop.table(A,1)*100,2)

###Accuray metric####

sum(diag(A))/sum(A)

###Plot ROC curve & find the AUC metric###

install.packages("pROC")

library(pROC)

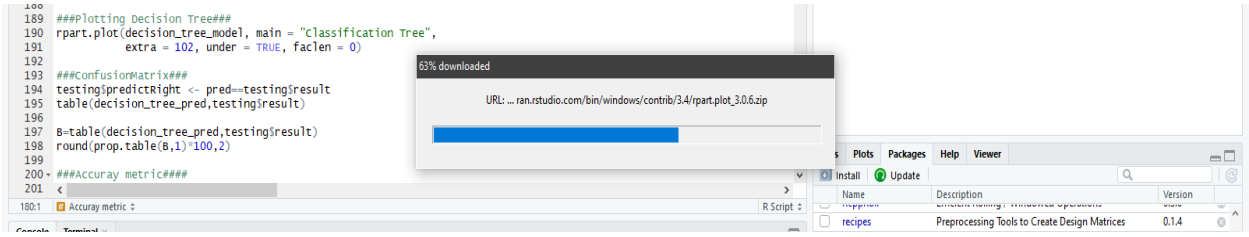
pred <-predict(modelrandom,testing,type= 'prob')

```
auc<-auc(testing$result,pred[,2])
auc
plot(roc(testing$result,pred[,2]))

###Finding the mtry###
bestmtry<-tuneRF(training, training$result, ntreeTry = 200, stepFactor = 1.2,improve= 0.01,
Trace=T , plot= T)
Bestmtry

#(2)DTs
###DecesionTree###
install.packages("rpart")
library(rpart)
```


```
install.packages("rpart.plot")
library(rpart.plot)
```




```
set.seed(1234)
decision_tree_model <- rpart(result ~ ., data = training, method = "class")
decision_tree_model

###Predictions###
decision_tree_pred <- predict(decision_tree_model,testing , type = "class")
decision_tree_pred
```

```
###Plotting DTs###
rpart.plot(decision_tree_model, main = "Classification Tree", extra = 102, under = TRUE, faclen = 0)

###ConfusionMatrix###
testing$predictRight <- pred==testing$result
table(decision_tree_pred,testing$result)
B=table(decision_tree_pred,testing$result)
round(prop.table(B,1)*100,2)

###Accuray metric####
sum(diag(B))/sum(B)

#(3)SVM
###supportVectorMachine###
install.packages("e1071")
library(e1071)
set.seed(1234)
Svm_model<-svm(result ~ .,data=training)
Svm_model
summary(Svm_model)

###Predictions###
Svm_pred <- predict(Svm_model,testing , type = "class")
Svm_pred

###ConfusionMatrix###
testing$predictRight <- pred==testing$result
table(Svm_pred,testing$result)
C=table(Svm_pred,testing$result)
round(prop.table(C,1)*100,2)

###Accuray metric####
sum(diag(C))/sum(C)
```

###Ensemble###

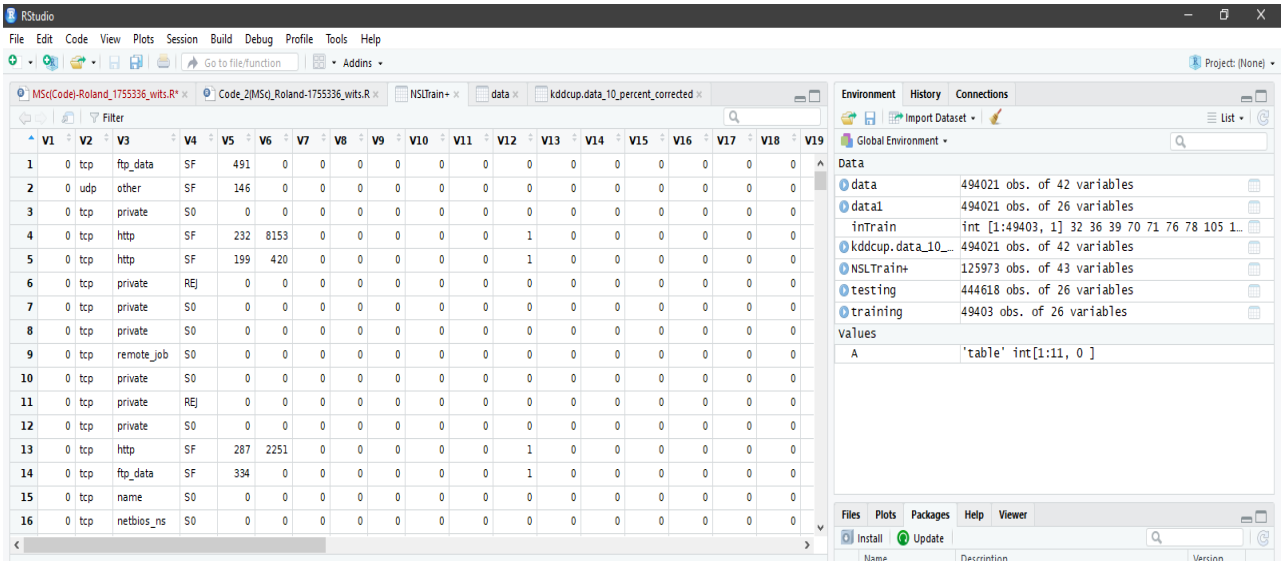
modelrandom<-randomForest(result ~ .,data=training, mtry=3, ntree=20)

decision_tree_model <- rpart(result ~ ., data = training, method = "class")

Svm_model<-svm(result ~ .,data=training)

(2)#NSL#datasets#

data<-read.table("C:/Users/1755336/Downloads/NSLTrain+.csv", sep = ",")



View (data)

nrow(data)

ncol(data)