DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

MODIFIED STACKING ENSEMBLE MACHINE LEARNING METHOD FOR NETWORK INTRUSION DETECTION

by
Necati DEMİR

June, 2018
İZMİR

MODIFIED STACKING ENSEMBLE MACHINE LEARNING METHOD FOR NETWORK INTRUSION DETECTION

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Computer Engineering

by
Necati DEMİR

June, 2018
İZMİR

Ph.D. THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"MODIFIED STACKING ENSEMBLE MACHINE LEARNING METHOD FOR NETWORK INTRUSION DETECTION"** completed by **NECATİ DEMİR** under supervision of **ASST. PROF. DR. GÖKHAN DALKILIÇ** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Doctor of Philosophy.

Asst. Prof. Dr. Gökhan DALKILIÇ

Supervisor

Asst. Prof. Dr. Tuğkan TUĞLULAR

Thesis Committee Member

Asst. Prof. Dr. Yavuz ŞENOL

Thesis Committee Member

Prof.Dr. Mehmet Ufuk Çoğlayan

Examining Committee Member

Assoc. Prof. Dr. Mehmet Yıldırım

Examining Committee Member

Prof.Dr. Latif SALUM

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENTS

Firstly, I would like to express my sincere gratitude to my advisor Asst. Prof. Dr. Gökhan DALKILIÇ for his support of my PhD study, for his patience and motivation. His guidance helped me to determine the correct path in my study.

Beside my advisor. I would like to thank to the members of my PhD thesis committee: Asst. Prof. Dr. Tuğkan TUĞLULAR and Asst. Prof. Dr. Yavuz ŞENOL for their comments and suggestions.

I would like to express my special gratitude to my wife Mehtap DEMİR for her support and patience during not only in the phase of writing my thesis but also in all steps of my PhD study. My daughter Doğa and my son Barış; your presences give me the strength in all aspects of my life.

Last, but not least, I would like to thank my parents, Metin and Sebahat.

Necati DEMİR

MODIFIED STACKING ENSEMBLE MACHINE LEARNING METHOD FOR NETWORK INTRUSION DETECTION

ABSTRACT

Machine learning (ML) methods became highly popular since the amount of the data produced on the Internet started increasing exponentially, although it was a known topic in academic studies before that era. It started becoming highly hard to extract rules from this huge amount of data or find patterns. ML started playing an important role in the area of finding patterns and extracting rules.

With increasing number of people accessing the Internet and having smart mobile phones, the possible threats in the Internet became important topic in network security studies. The conventional way of detection network intrusion is to use signature based rules (pre-defined rules), where this type of rules can't detect unknown signatures even though the type of the attack is the same. In last decade, ML started to be used in network security studies more often.

This study is based on using stacking ensemble machine learning method for the purpose of detecting network intrusion. In this study, we propose two different methods to improve the performance of ML methods to detect network intrusion.

Firstly, we used different combination algorithms and different base model selection methods to improve the performance of stacking ensemble method which provided significant results when it is compared to conventional machine learning methods. Secondly, we used genetic algorithm to for the base model selection phase of the first study.

Keywords: Machine learning, ensemble methods, stacking ensemble, genetic algorithm, logistic regression, decision tree, naïve bayes, oversampling, intrusion detection

AĞ İHLALİNİN TESPİTİ İÇİN MODİFİYE EDİLMİŞ İSTİFLEME TOPLULUK MAKİNE ÖĞRENME TEKNİĞİ

ÖZ

Makine öğrenmesi (MÖ) internet uzerindeki veri boyutlarının eksponansiyel olarak artması sonucunda, akademik çalışmalarda hali hazırda zaten biliniyor olmasına rağmen, yüksek bir popülerliğe sahip olmaya başladı. Bunla beraber, bu yüksek hacimli veriden herhangi bir kural çıkartmak ya da örüntüler bulmak gittikçe zorlaşmaya başladı. MÖ örüntü bulma ve kuralların otomatik olarak çıkarılması alanında önemli bir rol oynamaya başladı.

Çok sayıda kişinin internete erişmesi ve akıllı cep telefonlara sahip olmasıyla beraber, internet üzerindeki olası tehditler ağ güvenliği çalışmalarında önemli bir konu olmaya başladı. Ağ saldırılarının tespitinde kullanılan geleneksel yöntem imza bazlı kuralların (önceden belirlenmiş kurallar) kullanılmasıdır, ki bu kurallar aynı saldırı tipinde olsa bile bilinmeyen imzaya sahip saldırıları tespit edemez. Son yıllarda, MÖ ağ güvenliği çalışmalarında daha sıkça kullanılmaya başlandı.

Bu çalışma ağ saldırılarının tespitinde kullanılması amacıyla *stacking ensemble* makine öğrenmesi yöntemin üzerine kurgulanmıştır. Bu çalışmada, ağ saldırılarının tespitinde MÖ yöntemlerinin performansını arttırmasıyla amacıyla iki yöntem önerilmiştir.

İlk olarak, istifleme topluluk (*stacking ensemble)* yönteminin performansını arttırmak amacıyla farklı birleştirme algoritmaları ve farklı taban model seçme yöntemleri kullandık ve geleneksel makine öğrenmesi yöntemleriyle karşılaştırınca başarılı sonuçlar aldık. İkincil olarak, bir önceki çalışmanin baz model seçme aşamasında genetik algoritma kullandık.

Anahtar kelimeler: Makine öğrenmesi, topluluk (*ensemble*) yöntemleri, istifleme topluluk (*stacking ensemble*), genetik algoritma, lojistik regresyon, karar ağacı, naïve bayes, yüksek örnekleme, saldırı tespiti

CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

CHAPTER ONE
INTRODUCTION

1.1 Background

1.1.1 Machine Learning

Arthur Samuel described machine learning as “the field of study that gives computers the ability to learn without being explicitly programmed” in 1959. We can divide the the machine learning methods into three categories to explain how the learning works;

- supervised learning,
- unsupervised learning,
- reinforcement learning.

This categorization is also shown in Figure 1.1. Although there are many other algorithms used in different studies and applications, this figure shows the algorithms which will be covered in this thesis. There are three algorithm names in Figure 1.1 with asterisk (*) near it. Those algorithms are described in section 2, in detail because those algorithms are the main base algorithms used in this thesis.

Supervised learning methods deals with the data given as X,y pairs. The goal of supervised learning is mapping from X to y. If the the variable y is a continuous(numeric) variable, this type of supervised algorithms are called regression algorithms; if the variable y is a categorical data, then those problems are solved with classification algorithms. As an example; “predicting the price of a house” is a regression problem and “detecting if an incoming e-mail is a spam or not” is a classification problem

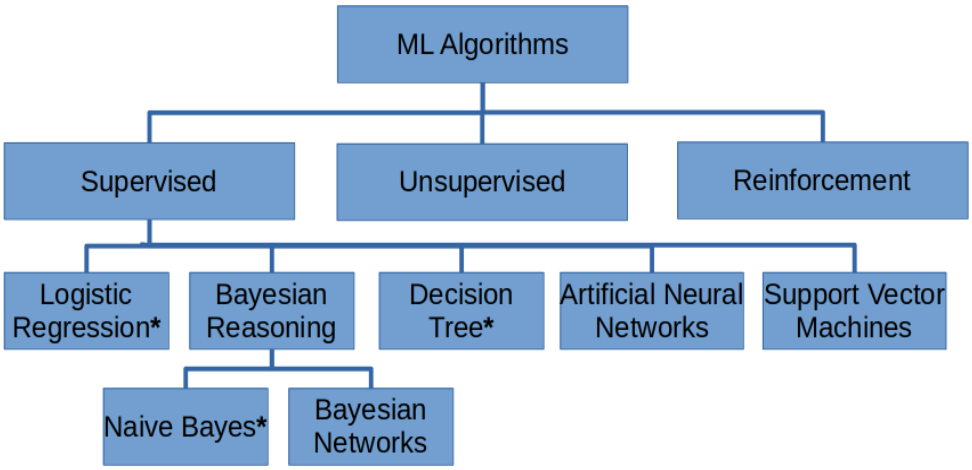

Figure 1.1 Categories of machine learning algorithms

Unlike supervised learning methods, unsupervised learning methods does not deal with X,y pair and they don't try to find a function to fit from X to y. The goal of unsupervised learning is to find the patterns in the dataset. For example; finding the patterns in the purchases of a retail store's customers can be shown as an example for unsupervised learning. Another use case for unsupervised learning is clustering where the goal is to group the similar items in a dataset. Grouping the similar customers by the recent purchased items is a perfect example for clustering.

Reinforcement learning is a method where it can be seen a combination of supervised and unsupervised learning. The reinforcement learning algorithms works on reward systems. The algorithm's inputs are a task to solve and a reward function. If the algorithm does a successful move, the reward function give positive score otherwise it gives a negative score. The algorithm starts learning based on this rewards. Reinforcement learning is used in robotics. As an example task; a robot can learn to navigate inside a maze based on given scores produced by reward function.

1.1.2 Intrusion Detection

Intrusion detection is defined as the process of detecting attacks and misuse of computer systems. Intrusion detection systems can be categorized into two categories by where the detection takes place:

- network intrusion detection systems,
- host based intrusion detection systems

In practice, a network intrusion detection system (NIDS) is a device located at strategic point in a network and monitors the all network activity. Even though it is mostly used to monitor the traffic of all network, it can also be used to monitor the network traffic of a single server. The role of a NIDS is to detect the attacks or unauthorized activities. In addition to monitoring the network traffic, NIDS can also take a role to analyze the log files of the servers. Although a NIDS has a protective role in a network, it can also take proactive role and scan the network to find the exploits.

The goal of a host based intrusion detection system (HIDS) is to find the suspicious activity in a single computer system and in practice, it is a software that runs on the operating system. HIDS can monitor the inbound/outbound network activity to detect the suspicious activities or it may take a snapshot of system files and then compare it to the original version to detect the suspicious activities.

Intrusion detection systems can also be categorized into two categories by how the detection works:

- Signature based intrusion detection
- Anomaly based intrusion detection

Signature based intrusion detection is a method where the goal is to detect attacks by known patterns. For example; anti-virus systems checks the byte sequence in a file to verify if a file contains a virus. The same method also can be used in network intrusion detection, where the goal of a NIDS is to search the byte sequences in the network traffic.

Anomaly based intrusion detection systems are used to detect non-normal behaviors. Generally, machine learning methods are used for this type of detection systems. Machine learning algorithms are trained to understand the normal behavior of the traffic and then any deviation from this behavior is marked as an anomaly.

1.1.3 Internet Protocol Suite

Internet protocol suite is a set of communication protocols used in computer networks and also known as TCP/IP because the foundational protocols are the Transmission Control Protocol (TCP) and the Internet Protocol (IP). This protocol suite contains the specifications for end-to-end communication on how the data is packed, addressed, transmitted, routed and received. It contains four abstract layers; link layer, internet layer, transport layer and application layer.

The communication between computers are done via packets that contain the data that will be delivered and the header that contains information about where it came from and where it will go and etc. The link layer is responsible to carry the packet over one link at a time.

Internet layer is responsible of sending packets across multiple networks and this process is called routing. This layer basically contains two functions; host identification via IP addressing system and packet routing.

Transport layer is used to establish the data channels between computers for data exchange. End-to-end communication in transport layer can be implemented in two ways: connection-oriented and connectionless. Connection-oriented channels are established via TCP and the connectionless channels are established via UDP.

Application layer includes the protocols that are used by applications. For example; Hypertext Transfer Protocol (HTTP) is an example of application layer protocol and web servers use HTTP which uses TCP. Data in the application layer is encapsulated into transport layer (example; TCP or UDP), and those encapsulated

packets are transmitted to lower layer protocols for the data transfer between computers.

Figure 1.2 shows the layers that we discussed above and also shows the protocols that are used in those layers.

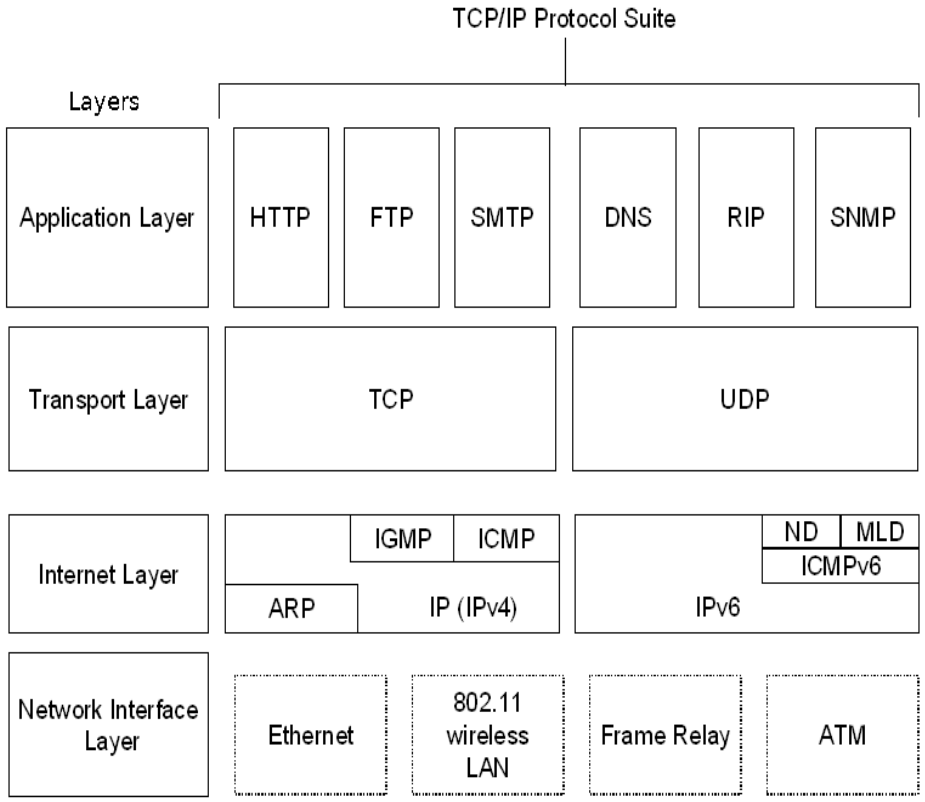

Figure 1.2 Internet Protocol Suite (Davis (2004))

The next section will discuss the KDD'99 dataset and this dataset contains the features extracted from network source so we will discuss the details of TCP and UDP headers.

TCP accepts data from a data stream, divides it into chunks and adds header to each chunk. A TCP header contains the following data:

- source port (16 bits)
- destination port (16 bits)

- sequence number (32 bits): if SYN flag is set, this field is used for initial sequence number otherwise it is used as the accumulated sequence number
- acknowledgment number (32 bits): if ACK flag is set, this field contains the sequence number that the sender of ACK sender is expecting
- data offset (4 bits): specifies the size of the TCP header
- reserved (3 bits): it is reserved for future use and should be set zero
- flag (9 bits): contains 9 flags each with 1 bit size:
    - NS: concealment protection
    - CWR: indicate that it received a TCP segment with the ECE flag set
    - ECE: if SYN is set, this flag indicates that the TCP peer has ECN capability, otherwise it indicates a network congestion.
    - URG: indicates the significancy
    - ACK: all packets after SYN packet sent, this should be set
    - PSH: this field is used to ask to push the buffer
    - RST: is used to reset the connection
    - SYN: only the first packet sent from each peer should have this flag
    - FIN: indicates that the packet is the last packet
- window size (16 bits)
- checksum (16 bits): this field is used for error-checking
- urgent pointer (16 bits): if the URG flag is set, this field is used to indicate the last urgent data byte
- options (variable 0-320 bits): the length of this field is variable depending on the data offset field.
- padding: this field is used to ensure that the size of the tcp head is on 32-bit boundary and this field is composed of zeros.

UDP header consists of only 4 headers:

- source port number:
- destination port number
- length: this fields is used to store the size of the UDP header and payload.
- checksum: this field is used for error-checking.

1.1.4 KDD '99 Dataset

We used KDD'99 dataset, which is widely used in "network intrusion detection" studies. The KDD'99 dataset was used for the 3rd International Knowledge Discovery and Data Mining Tools Competition. The competitors were given a task and the task was to build a an algorithm which detects network intrusion placed in this dataset. This dataset was acquired from 1998 DARPA intrusion detection evaluation program. DARPA set up an environment to collect raw TCP/IP data for a local area network (LAN) that simulates a typical U.S. Air Force LAN, and that LAN was operated as if it was a true environment, and it was blasted with multiple attacks.

The attacks in this dataset are divided into 4 different categories:

- *Probe:* The goal of the attacker is to get information about the target host.
- *Denial of Service (DoS):* The goal of the attacker is to prevent normal users from using a service. This can be accomplished by two methods;
  - keeping the resources of the target host busy by launching excessive number of normal communication transactions, which is a widely used method,
  - exploiting the services.
- *User to Root (u2r):* Attacker can access to target machine and his goal is to gain root privileges.
- *Remote to Local (r2l):* Attacker cannot access to the target host and his goal is to access to the local system.

Each attack category contains more than one attack type and those are listed in Table 1.1. Some attack types are seen with an asterisk(*) sign near it which indicates that those attack types do not exist in training dataset but exist in test dataset.

Table 1.1 Attack types by categories

Attack Category	Attack Type
Probe	Ipsweep, mscan*, nmap, portsweep, saint*, satan
Denial of Service (DoS)	Apache2*, back, land, mailbomb*, neptune, pod, processtable*, smurf, teardrop, udpstorm*
User to Root (u2r)	buffer_overflow, loadmodule, perl, ps*, rootkit, sqlattack*, xterm*
Remote to Local (r2l)	ftp_write, guess_passwd, httptunnel*, imap, multihop, named*, phf, sendmail*, snmpgetattack*, snmpguess*, spy, warezclient, warezmaster, worm*, xlock*, xsnoop*

In 1999, DARPA dataset was revised to create KDD'99 dataset that contains 265 summarized information of TCP connections, not TCP dumps. Each connection is summarized with 41 features, that can be grouped in 4 categories:

- *Basic features:* Basic features are extracted from headers (section 1.1.3) and the payload is not investigated during the extraction of those features. The first 9 features in Table A-1.1 of Appendix-1 are the basic features. Duration is the length of connection in seconds, protocol_type is the type of the protocol (tcp, udp), service is the service (http, telnet, …), src_bytes is the number of data bytes from source to destination, dst_bytes is the number of data bytes from destination to source, flag is the status of the connection, land indicates if the connection from/to same host/port, wrong_fragment is the number of wrong fragments and urgent is the number of urgent packets. Those fields are extracted by using only header fields that we discussed in section 1.1.3.
- *Content features:* Content features are extracted from payload and domain knowledge is needed to extract features.
- *Time-based traffic features:* These features are computed features over a 2 seconds temporal window. For example, error rate is a time-based traffic feature, which describes the percentage of connections having SYN errors.

- *Host-based traffic features:* These features are computed features over a 100 connections temporal window. Some probing attacks scan the hosts where computing these features gives insight.

Table 1.2 Components of KDD'99 dataset

Dataset	Probe	DoS	U2R	R2L	Normal
10% of KDD	4,107 (0.83%)	391,458 (79.24%)	52 (0.01%)	1,126 (0.22%)	97,277 (19.69%)
Corrected KDD	4,166 (1.34%)	229,853 (73.90%)	70 (0.02%)	16,347 (5.26%)	60,593 (19.48%)
Whole KDD	41,102 (0.84%)	3,883,370 (79.28%)	52 (0.001%)	1,126 (0.23%)	972,780 (19.86%)

The KDD'99 dataset consists of three components as seen in Table 1.2 and all these components are extracted from the same source. Figures 1.3-1.5 show the distribution of the labeled categories on each component of KDD dataset. In the International Knowledge Discovery and Data Mining Tools Competition ’99, only 10% of KDD, which contains 22 attack types, was used for training as in our study. Corrected KDD dataset contains different statistical distributions for attacks and also contains 14 additional attacks. We used corrected KDD dataset for testing purposes.

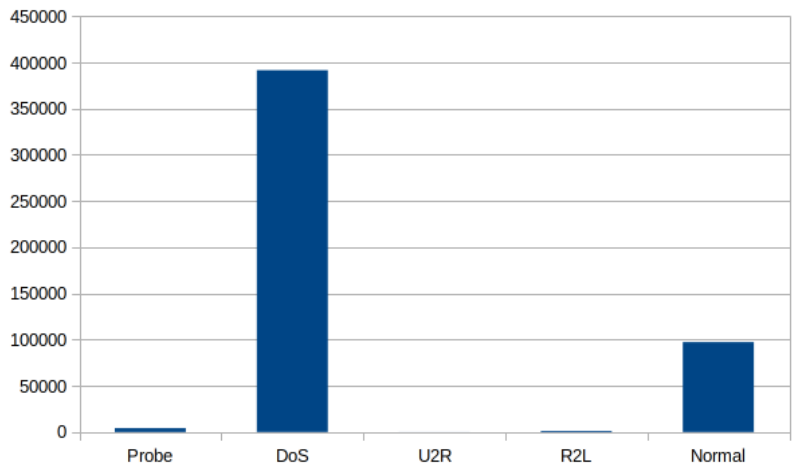

Figure 1.3 The distribution of labeled categories on 10% of KDD

Figures 1.3, 1.4 and 1.5 show the distribution of labeled categories for 10% of KDD, corrected KDD and whole KDD respectively. Major labels are dos and normal labeled categories which means that those categories has high number of instances in the datasets.

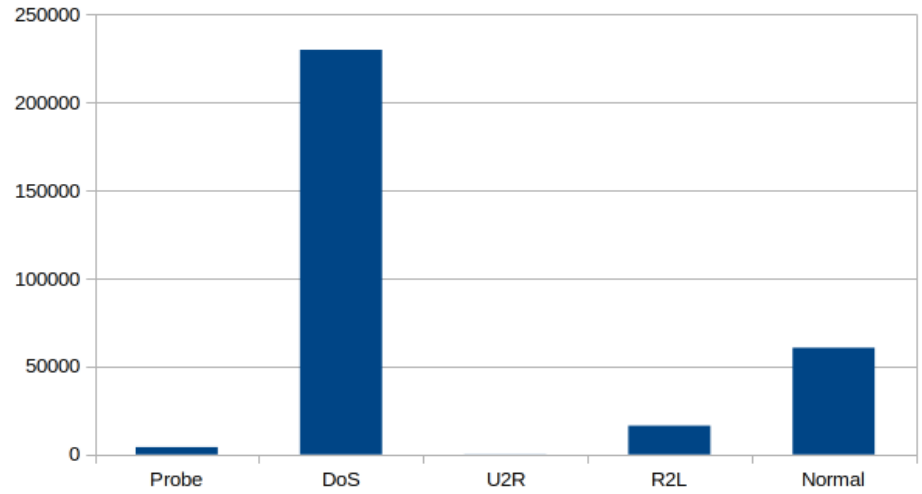

Figure 1.4 The distribution of labeled categories on corrected KDD

It is also important to mention that while 0.22% of the training dataset (10% of KDD) is r2l labeled data, 5.26% of the testing data (corrected KDD) is r2l labeled data; which challenges the r2l predictions.

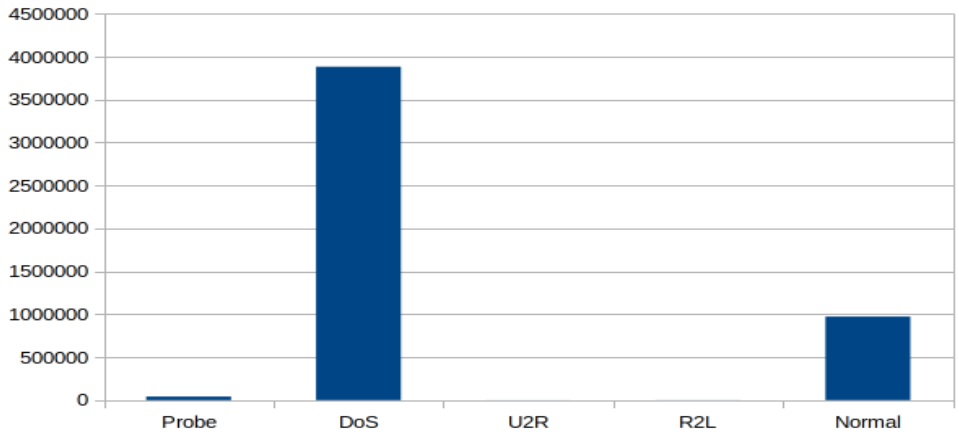

Figure 1.5 The distribution of labeled categories on whole KDD

Appendix-1 contains of KDD'99 dataset. Appendix-1 contains 3 tables; Table A-1.1 contains the type (numerical or categorical) of each 41 features, Table A-1.2 and Table A-1.3 contain the first 5 rows of 10% of KDD and corrected KDD datasets, respectively.

Even though we decided to use KDD'99 dataset for this study, there are various other datasets that can be used for machine learning studies, and the main advantage of KDD'99 is it is the most used dataset in academic studies. Özgür & Erdem (2016) studied the papers which used machine learning for intrusion detection between 2010 and 2015 and figured out that KDD'99 dataset is the most used dataset. According to this study by Özgür & Erdem (2016), KDD'99 dataset was used in 133 papers while the second most used dataset, NSL-KDD, was used in 23 papers and the third most used dataset is DARPA which is used only in 9 papers. DARPA dataset is a raw dataset, the researcher needs to focus on extracting the features. NSL-KDD dataset is a re-sampled version of KDD'99 dataset and the duplicated records are removed from KDD'99 dataset. The main advantage of KDD'99 dataset to DARPA dataset is having features which helps the researcher to focus on the machine learning part of the study. The main advantage of KDD'99 dataset to NSL-KDD dataset is being used more and this helps the researchers to see and evaluate their studies precisely when KDD'99 is used.

1.2 Motivation and Contribution

1.2.1 Motivation

With the increasing usage of Internet in the last 20 years, the security became one of the main issues. This problem also increased the demand of solutions for intrusion detection. Both the private sector and the academic world focused on solutions.

Academic studies are mostly focused on finding the attacks that are not known, because the signature based solutions is already providing a solution to find the known attacks. Those academic studies are using the power of machine learning for

this purpose. As we mentioned section 1.1.1, the classification methods can be used to find the attacks. Since the nature of attacks are complicated, those methods does not provide high accuracy in detecting attacks. Ensemble machine learning methods started playing a role in intrusion detection studies to achieve higher accuracy.

We will discuss the details for ensemble methods in section 4 but we can briefly explain that the goal of ensemble methods. The goal of ensemble methods is to combine more than one machine learning model to create a new model. Ensemble is an umbrella term for those methods that combine multiple models, so we need to say that we used an ensemble method which is called "stacked generalization (stacking)" for detecting network intrusion by using KDD'99 dataset.

In this thesis, our motivation for this study is driven by the following:

• Although there are various studies applying ensemble machine learning techniques, to our best knowledge, there is no study that compares linear regression, decision tree, and Naïve Bayes when they are used for a combiner method in stacking.

• To our best knowledge, there is no study applying stacking ensemble on KDD'99 dataset, which uses various selection methods to be given as input to the combiner algorithm.

1.2.2 Contribution

As we discussed in motivation section, our motivation is based on using stacking machine learning method for the task of network intrusion detection. Although we will discuss stacking ensemble method in section 4.5 and will describe how we used stacking in section 5.1, we need to give a brief summary of how stacking works to explain our contribution.

Stacking is an ensemble method and it consists of two phases; where the in first level, machine learning models are generated and in the second level those the

outputs of machine learning models are combined by another machine learning model to generate the final output.

Our contribution is on both levels. In first level, we used random selection method to generate dataset and those datasets are used for generating models. We generated 100 models but we needed to select the best models and we used various metric (explain in section 5.1) to select the best 10 models. In second level, we used 3 different machine learning algorithms to combine the first level models. Our paper, which contains the contribution mentioned in this paragraph, was accepted by *Turkish Journal Of Electrical Engineering & Computer Sciences* with the title *Modifed stacking ensemble approach to detect network intrusion* (Demir & Dalkılıç, 2018).

The second contribution we made is using genetic algorithm to select the models based on multi objective optimization. So, the second contribution can be seen an improvement of first contribution which tries to maximize all true positive rates.

1.3 Thesis Organization

This thesis contains seven chapters. In this chapter, we started by defining the main concepts that we will use through this study. We discussed what machine learning is, what intrusion detection is and explained the dataset we will use. After this brief explanation, we provided our motivation and contribution.

Chapter 2 contains the detailed descriptions of the ML concepts that we will use. We explained the classification algorithms and the data pre-processing methods that we will use in this study.

Chapter 3 contains the previous academic studies related to this study, which is basically using machine learning techniques to detect network intrusion. We

segmented the sub sections by algorithm names to help the reader to read the section easier.

Chapter 4 contains a detailed description of ensemble methods, which is the method we use in this study. This chapter not only describes the method (stacking ensemble method) we use, but also explains the other methods that fall under the term ensemble methods.

In chapter 5, we begin by explaining our approach (parameterized stacking ensemble method) which is based on stacking ensemble method. After this explanation, we describe how we conduct experiments and discuss the results of the experiments.

Chapter 6 contains the details of the study which is the improved version of the study discussed in chapter 5. We used genetic algorithm for the base model selection phase of the previous study.

In chapter 7, we presented our conclusions and the possible future studies that can be implemented based on this study.

CHAPTER TWO
MACHINE LEARNING

Machine learning is a field where the goal is teach the computer to understand the patterns in the data without being programmed. We gave a very brief explanation about machine learning in section 1.1.1, and discussed that the classification methods can be used for the task of network intrusion detection. In this chapter, we will explain the classification algorithms that are used in this study.

2.1 Supervised Learning

2.1.1 Logistic Regression

Logistic regression is a function which calculates the probability of a discrete outcome based on the given data. Logistic regression is based on linear regression and logit function is applied to the output of linear regression to find the probability. Linear regression is defined by the given equation:

$$w_1 x_1 + w_2 x_2 + \ldots = \sum w_i x_i \tag{2.1}$$

In Equation (2.1), x denotes the given data and the w denotes the weights. A logit function is formulated in Equation (2.2) and logit function has the ability to convert any real value to a value between 0 and 1. As an example; Figure 2.1 shows the the output of logit function when it takes real values between -10 and 10.

$$logit(z) = \frac{1}{(1 + e^{-z})} \tag{2.2}$$

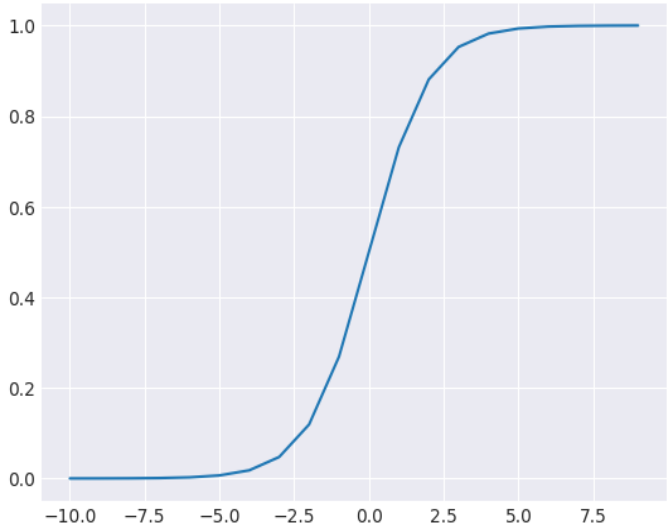

Figure 2.1 Logit function

We can apply logit function for linear regression to find the the logistic regression equation, the final equation for logistic regression is show in Equation (2.3).

$$f(x)=\frac{1}{1-e^{-\sum w_i \cdot x_i}} \tag{2.3}$$

2.1.2 Decision Tree

Decision tree is used for approximating discrete-valued functions and the learned function can be represented as a tree diagram or can be expressed as if-else rules which makes it a human-readable solution. Figure 2.2 shows an example decision tree that shows if a person will go to play tennis outside based on her historical behavior (Mitchell, 1997) We will use this popular example, which was used by Mitchell (1997), in this section to explain how a decision tree is constructed.

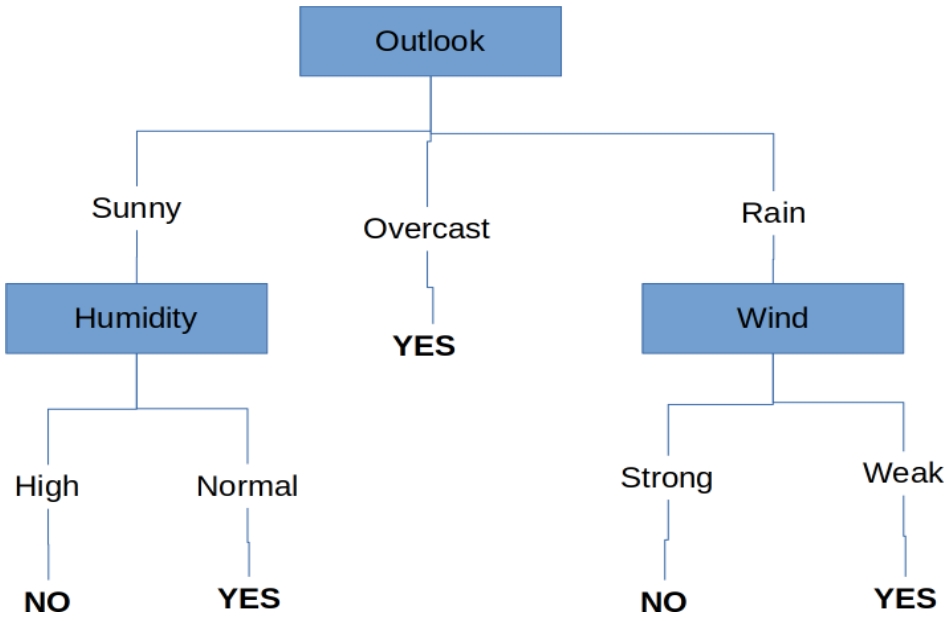

Figure 2.2 An example decision tree

Figure 2.2 shows a decision tree which can be easily expressed with if-else rules:

- If outlook is sunny and humidity is high, she will not play tennis,
- If outlook is sunny and humidity is normal, she will play tennis,
- If outlook is overcast, she will play tennis
- If outlook is rainy and the wind is strong, she will not play tennis,
- If outlook is rainy and the wind is weak, she will play tennis.

There are several implementations of decision tree algorithm, in this section we will explain ID3. ID3 starts constructing the tree by finding which attribute will be used as root node. And then, it continues finding which attribute is the best to split the decision tree in each node. ID3 uses information gain criteria to decide the attribute. We need to explain the entropy before continuing with information gain. Entropy is used to calculate the impurity of a set of examples and is formulated in Equation (2.4).

$$Entropy(x) = \sum_{i=2}^{n} -p_i \log_2 p_i \tag{2.4}$$

As an example; if a set of examples has 9 positive and 5 negative examples in a binary classification problem, the entropy is 0.94 and the calculation is given in Equation (2.5).

$$Entropy = -9/14 \log_2 9/14 - 5/14 \log_2 5/14 = 0.94 \tag{2.5}$$

If all examples are in same class, the entropy will be 0. The entropy can be between 0 and $\log_2(n)$. In previous example n=2.

Given entropy formula, we can explain information gain. Information gain is the difference between two entropies. The information gain is calculated by the given formula given in Equation (2.6).

$$InformationGain(x) = InitialEntropy - \sum_{i=1}^{v} p(x_i)\, Entropy(x_i) \tag{2.6}$$

We will use example shown in Figure 2.2 and in the Table 2.1. As an example; we can calculate the information gain for the attribute "outlook". First step is calculating the entropy of the dataset where it contains 9 positive and 5 negative examples; we did this calculation in Equation (2.5), so we know that the entropy is 0.94. The next step is calculating the entropies for each value of outlook attribute and multiple by the probability of that value. Outlook attribute has 3 values: sunny, overcast, rain; we will sum those entropies and subtract it from 0.94. Those calculations are done in Equations (2.7-2.10).

$$Entropy(outlook = sunny) \Rightarrow$$
$$\Rightarrow -(3/5)\log 2(3/5) - (2/5)\log 2(2/5) = 0.97 \tag{2.7}$$

$$Entropy(outlook = overcast) \Rightarrow$$
$$\Rightarrow -(0/4) * \log 2(0/4) - (4/4)\log 2(4/4) = 0 \tag{2.8}$$

$$Entropy(outlook = rain) \Rightarrow$$
$$\Rightarrow -(2/5) * \log 2(2/5) - (3/5) * \log 2(3/5) = 0.97 \tag{2.9}$$

$$InformationGain(outlook) \Rightarrow$$
$$\Rightarrow 0.94 - ((5/14) * 0.97 + (4/14) * 0 + (5/14) * 0.97) = 0.247 \tag{2.10}$$

Table 2.1 An example dataset (Mitchell, 1997)

Outlook	Humidity	Temperature	Wind	Play Tennis
sunny	high	hot	weak	no
sunny	high	hot	strong	no
overcast	high	hot	weak	yes
rain	high	mild	weak	yes
rain	normal	cool	weak	yes
rain	normal	cool	strong	no
overcast	normal	cool	strong	yes
sunny	high	mild	weak	no
sunny	normal	cool	weak	yes
rain	normal	mild	weak	yes
sunny	normal	mild	strong	yes
overcast	high	mild	strong	yes
overcast	normal	hot	weak	yes
rain	high	mild	strong	no

Equation (2.10) provides the final results for the information gain of outlook attribute. We can do the same calculations for each attribute to find the root node of the decision tree. When those calculations are done; we will find the results below:

- InformationGain(Outlook) = 0.247
- InformationGain(Humidity) = 0.152
- InformationGain(Wind) = 0.049
- InformationGain(Temperature) = 0.029

Outlook attribute provides the highest information gain, so it can be used as the root node of the decision tree. For splitting the sub-level nodes, the same formulas will be used but instead of using the full data set, the sub datasets will be used.

If we use outlook attribute as the root node, there are three possible branches:

- outlook=rain
- outlook=sunny
- outlook=overcast

For splitting those three nodes, we need to find the best attribute again for each nodes. But, this time, we need to use the dataset where outlook is “rain” only, outlook is “sunny” only and outlook is ”overcast” only.

2.1.3 Naïve Bayes

Naïve Bayes algorithm is based on Bayes theorem and Bayes theorem is used to calculate the posterior probability of a hypothesis based on the prior probability. The prior probability is defined as the probability of observing data given the hypothesis. Bayes theorem is shown on Equation (2.11).

$$P(y|X) = \frac{P(X|y)\,P(y)}{P(X)} \tag{2.11}$$

P(y) denotes the probability of the outcome, P(X) denotes the probability of the data that will be observed, P(X|y) denotes the probability of the data observed given in the world where the outcome is y and P(y|X) defines the probability of the outcome based on given observed data

When Naïve Bayes is built on Bayes Theorem to implement a classifier, the goal is to find the probabilities of each possible outcome when X is given. Since X is constant, we can ignore the denominator P(X). So, the final formula becomes as seen on Equation (2.12).

$$f(X) = argmax_{y \in Y}\left(P(X|y)\,P(y)\right), \quad \textit{where } Y \textit{ is the set of all possible outcomes} \tag{2.12}$$

We can explain the Bayes Theorem using the Table 2.1. In this example; we will assume that this table consist of only two columns; “Outlook” and “Play Tennis”. “Outlook” will be used for **X** (given data) and “Play Tennis” will be used for **y**. We will calculate if he will play tennis or not when the outlook is sunny.

Equation (2.13) is used to calculate P(Play Tennis = yes| Outlook = sunny) and Equation (2.14) is used to calculate P(Play Tennis = no| Outlook = sunny).

$$\begin{gathered} P(Play\ Tennis = yes | Outlook = sunny) \Rightarrow \\ \Rightarrow P(Outlook = sunny | Play\ Tennis = yes) P(Play\ Tennis = yes) \\ \Rightarrow 2/9 * 9/14 = 0.14 \end{gathered} \tag{2.13}$$

$$\begin{gathered} P(Play\ Tennis = no | Outlook = sunny) \\ \Rightarrow P(Outlook = sunny | Play\ Tennis = no) P(Play\ Tennis = no) \\ \Rightarrow 3/5 * 5/14 = 0.21 \end{gathered} \tag{2.14}$$

After the previous calculations, since 0.21 is bigger than 0.14, Naïve Bayes gives the answer that he will not play tennis. This example used only one feature to calculate the results. The formula given on Equation (2.15).

$$f(X) = argmax_{y \in Y} P(y) \prod_{i=1}^{n} P(x_i | y) \tag{2.15}$$

2.1.4 Genetic Algorithm

Genetic algorithm (GA) is an optimization algorithm inspired by biological evolution and it involves operations like mutation, crossover and selection which are also inspired from biological evolution.

Even though, GA is an optimization method and can be used for various reasons, in this study we will use for classification problem so we will describe GA in this section.

Genetic algorithm evolves a population of candidate solutions towards a better solution, based on given fitness function. Each candidate (also called individual) has features called chromosomes in the context of genetic algorithm. In each iteration, those chromosomes are mutated or crossover, new candidates are generated or some candidates are dropped off using the given fitness function. The iteration process terminates when the number of iterations reach a defined maximum value, or when a satisfactory fitness level is reached.

Figure 2.3 provides the process of GA, we will describe each step one by one in the rest of the section.

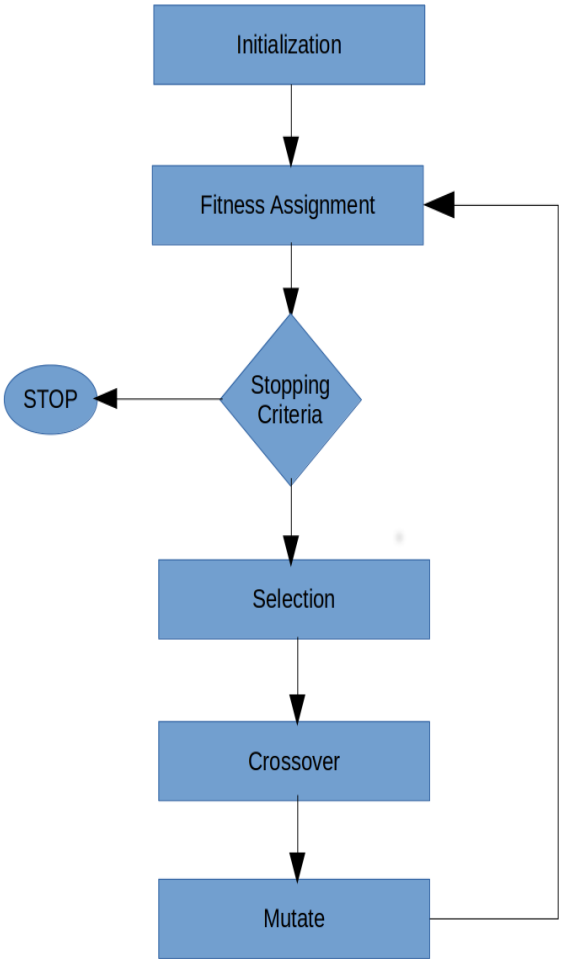

Figure 2.3 The process of GA

- Initialization:
  - This step consists of randomly generation of the initial population. We discussed that each individual in the population has features called chromosomes. We can see a sample dataset in Table 2.2 which contains 4 features (chromosomes) and 4 individuals.

Table 2.2 An example dataset for the initialization step of GA

	c#1	**c#2**	**c#3**	**c#4**
individual#1	1	0	1	1
individual#2	1	1	0	0
individual#3	1	1	0	0
individual#4	0	0	1	0

- Fitness Assignment:
  - The second step of GA is to assign fitness values to individuals. Fitness value is the output of a function which will be maximized or minimized. We will continue using the same example in Table 2.2 and we will use the sum of bits as a fitness function and we will try to maximize the fitness function in this example.

$$fitness(individual) = \sum_{i=1}^{4} individual_{c\#i} \tag{2.16}$$

  - Equation (2.16) shows the fitness function for this example. After calculation the fitness function, the results can be seen in Table 2.3.

Table 2.3 An example dataset for the fitness assignment step of GA

	c#1	**c#2**	**c#3**	**c#4**	**fitness**
individual#1	1	0	1	1	3
individual#2	1	1	0	1	2
individual#3	1	1	0	0	2
individual#4	0	0	1	0	1

- Stopping Criteria:
  - The evaluation step comes after the fitness assignment step, so the GA loop can decide to stop or continue. Stopping criteria can be a value which shows the maximum number of iterations, or it can be a value which is a satisfactory fitness value defined by the user.

- Selection:
  - This step of GA is used to keep a portion of individuals based on the given fitness function. In this example, the goal is to maximize the fitness function and we will select the half of the population for next steps (individual#1 and individual#2).
- Crossover:
  - This step is used to generate new child individuals by combining the chromosomes of the selected individuals. In this example; we will crossover individual#1 and individual#2 and will generate individual#5 and individual#6.
  - Individual#5 inherits c#1 and c#2 from individual#1 and inherits c#3 and c#4 from individual#2. Individual#6 inherits c#1 and c#2 from individual#2 and inherits c#3 and c#4 from individual#1. Table 2.4 shows the status of the population after the crossover step.

Table 2.4 An example dataset for the crossover step of GA

	c#1	**c#2**	**c#3**	**c#4**
individual#1	1	0	1	1
individual#2	1	1	0	1
individual#5	1	0	0	1
individual#6	1	1	1	1

- Mutate:
  - This step is used to manipulate the chromosomes based on given probability. In this example, we will assume that the c#3 of individual#1 and individual#2 is mutated which means they are converted to 1 and 0, respectively. The status of population can be seen in Table 2.5.

Table 2.5 An example dataset for the mutate step of GA

	c#1	**c#2**	**c#3**	**c#4**
individual#1	1	0	0	1
individual#2	1	1	1	1
individual#5	1	0	0	1
individual#6	1	1	1	1

- This is the latest step of the loop and the loop starts again from the step of fitness assignment.

We described the steps of genetic algorithm where the objective function has only one value. Genetic algorithm can also be used for an objective function which returns more than one value, which is called multi-objective optimization. The part that changes in this process is the selection step. We will explain this process by using an example. Figure 2.4 shows an example data set where each dot represents an individual and v1 and v2 (x and y axes) represent the the value of fitness values (which is the result of objective function). In summary, v1 and v2 are the output values of the objective function. In this example; the goal of the GA is to minimize the output values of the objective function.

There are 5 dots marked with the character *X,* which are selected in the selection step of the multi-objective GA process.

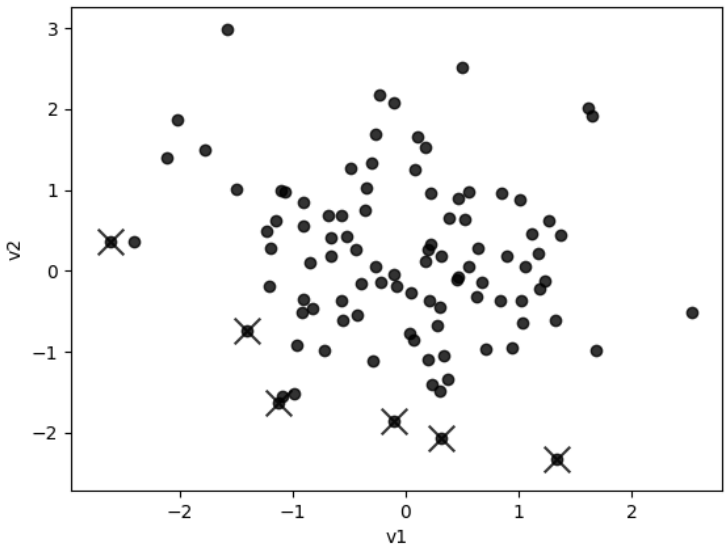

Figure 2.4 Example dataset for multi-objective GA

The algorithm starts with assuming that all of the individuals will be selected and then starts removing them for the list of selected variables. The pseudo-code can be seen on Figure 2.5.

```
def select(costs):
  #costs is a list of lists where each list contains the fitness values
  #example -> costs = [ [0,1], [1,2] ]

  #initialize the array of selected with TRUEs
  selected = [True for i in range(len(costs))]

  for element in costs:
    v1, v2 = element
    for ix,_element in enumerate(costs):
      if selected[ix] == True:
        _v1, _v2 = _element
        if _v1<=v1 or _v2<=v2:
          selected[ix] = True
        else:
          selected[ix] = False

  return selected
```

Figure 2.5 Pseudo-code of selection process for multi-objective GA

We can explain this process with a simple dataset shown in Table 2.6, where this dataset contains only 4 individuals and 2 fitness values.

Table 2.6 An example dataset for the multi-objective GA

individual#1	0.4	0.2
individual#2	0.2	0.3
individual#3	0.15	0.5
individual#4	0.25	0.35

- In the first step, we assume that all of the individuals will be selected: individual#1: True, individual#2: True, individual#3: True, individual#4: True.
- In the second step, we select individual#1 as a benchmark point and compare the others against it.
  - When we compare individual#1 and individual#2, individual#1's first fitness value is greater than individual#2's first fitness value, so there is still chance individual#2 can be selected, so we keep it in the list.
  - The same rule applies for individual#3 and individual#4, the final list is:
    - individual#1: True, individual#2: True, individual#3: True, individual#4: True.
- In the third step, we select individual#2 as the benchmark point and compare the others against it.
  - When we compare individual#1 to individual#2, we can see that second fitness value is smaller so we keep individual#1 in the list.
  - When we compare individual#3 to individual#2, we can see that first fitness value is smaller so we keep individual#3 in the list.
  - When we compare individual#4 to individual#2, we can see that first and second fitness values are greater, so we don't keep in the list any more because we are sure that the two fitness values are greater than one of the individuals.
  - So, the final list becomes: individual#1: True, individual#2: True, individual#3: True, individual#4: False.
- In the fourth and the fifth steps, the list does not change and the final list becomes:

- individual#1: True, individual#2: True, individual#3: True, individual#4:False

2.2 Data Preprocessing

Data preprocessing is the first step of the process of a machine learning based study after collecting the data. Data preprocessing is used to transform the raw data to a structure that machine learning algorithms can handle and which also helps to improve the quality of the model.

2.2.1 Normalization

Normalization is a method where the values are adjusted to a range, which is usually the range between 0 and 1. Even though some algorithms like decision tree can handle non-normalized data, some algorithms like logistic regression can't handle. For example; if the data contains age and height variables and age can take a value between 0 and 70 and height can take a value between 0.5 meters and 2 meters; logistic regression algorithm will give more weight to age.

$$X_{i_{new}} = \frac{X_i - X_{min}}{X_{max} - X_{min}} (max_{new} - min_{new}) + min_{new} \tag{2.17}$$

Equation (2.17) shows the formula to normalize the all elements of given X and transforms to a range between min_{new} and max_{new}. In this equation, X_{min} denotes the the minimum value of X and X_{max} denotes the maximum value of X. This formula is simplified as shown in Equation (2.18) when the the we want to transform the values to range between 0 and 1.

$$X_{i_{new}} = \frac{X_i - X_{min}}{X_{max} - X_{min}} \tag{2.18}$$

2.2.2 One-Hot Encoding

One-hot encoding is used to convert categorical data to numerical data which can be used as input for machine learning algorithms. Let's assume we have a set of variables as shown in Table 2.7 which shows the gender a small sample.

Table 2.7 An example dataset which shows the gender a small sample

Gender
Female
Female
Male
Female

When encoding is applied, Table 2.7 is transformed to Table 2.8.

Table 2.8 One-hot encoding applied to Table 2.7

Gender#Female	**Gender#Male**
1	0
1	0
0	1
1	0

2.2.3 Oversampling

Oversampling is used when dataset is not balanced. In section 1.1.4, we explained KDD'99 dataset which is used in this study and we explained that this dataset is not balanced per label. In this section, we will explain Synthetic Minority Oversampling Technique (SMOTE) oversampling method.

SMOTE is a method, which over-samples minority classes by creating synthetic examples. SMOTE method finds the nearest neighbor of each instance in that minority group and creates a synthetic example randomly between the examples.

Let's assume that the point (7,5) belongs to a minor class and its nearest neighbor is (5,4) which is also a minor class. The new data point can be calculated as seen in Equation (2.19).

$$\begin{aligned}(x1_{new}, x2_{new}) &= (7,5) + (rand(0,1) * (5-7), rand(0,1) * (4-5)) \\ &\Rightarrow (7,5) + (rand(0,1) * -2, rand(0,1) * -1) \\ &\Rightarrow (7,5) + rand(0,1) * (-2,-1)\end{aligned} \tag{2.19}$$

We showed how to create a synthetic example in Equation (2.19) and we used only 1 nearest neighbor. Depending on the required size of synthetic data, we could use more than one nearest neighbors.

As an example; if we have 100 instances in a minor class and if we want to create 200 synthetic data point, we can use 2 nearest neighbors and iterate the process for all instances. This process would end up creating 200 synthetic data points.

```
def over_sample(minor_class_data_points, n):

   new_data_points = []
   n_nearest_neighbor_per_data_point = n/len(minor_class_data_points)

   for data_point in minor_class_data_points:
      neigbors = find_nearest_neighbour(data_point, n_nearest_neighbor_per_data_point)
      for neighbor in neigbors:
         new_data_point = data_point + (neighbor-data_point)*rand(0,1)
         new_data_points.append( new_data_point )

   return new_data_points
```

Figure 2.6 SMOTE pseudo-code

Figure 2.6 shows the pseudo-code of a function which executes SMOTE oversampling method. This function uses the data points in minor class and the variable n which denotes the number of new data points to be created.

2.3 Metrics

In this sub section, we will discuss some metrics that are used to evaluate the implemented methods of this PhD study. Before proceeding this section, we need to explain what the confusion matrix is because it will be used in formulas described in this section.

Table 2.9 shows the structure of a confusion matrix, which will be used in this study. The sum of each row contains the actual corresponding labeled data. As an example, the sum of X_{00} , X_{01} , X_{02} , X_{03} , and X_{04} equals the total number of actual normal labeled data, and X_{00} shows the number of data labeled correctly by the model.

Table 2.9 An example of a confusion matrix

		Predicted				
		Normal	Probe	DoS	U2R	R2L
Actual	Normal	X_{00}	X_{01}	X_{02}	X_{03}	X_{04}
	Probe	X_{10}	X_{11}	X_{12}	X_{13}	X_{14}
	DoS	X_{20}	X_{21}	X_{22}	X_{23}	X_{24}
	U2R	X_{30}	X_{31}	X_{32}	X_{33}	X_{34}
	R2L	X_{40}	X_{41}	X_{42}	X_{43}	X_{44}

Accuracy: The accuracy is calculated using Equation (2.20) (Zhou, 2012) by dividing the total number of correctly classified samples by the total number of samples.

$$accuracy = \frac{x_{00} + x_{11} + x_{22} + x_{33} + x_{44}}{\sum(X)} \tag{2.20}$$

Although accuracy is a widely used metric, it can misguide the researcher when the data are unbalanced. As an example, let us assume there are two classes (class a and class b) and 90 of them belong to class a and 10 of them belong to class b. If the model predicts all of them as class a, the accuracy score will be 90% Although it detects none of the data labeled as class b.

Recall (true positive rate) mean: Recall mean also known as true positive rate mean is calculated as the mean of the recall value for each label. Recall mean formula is calculated using Equation (2.21) and the formula for each label is calculated using Equations (2.22-2.26). Recall is calculated by dividing the number of correctly classified samples by the number of true labeled samples in the original dataset.

$$recall_{mean} = \frac{\left(recall_{normal} + recall_{probe} + recall_{dos} + recall_{u2r} + recall_{r2l}\right)}{5} \tag{2.21}$$

$$recall_{normal} = \frac{X_{00}}{\sum x_{0i}} \tag{2.22}$$

$$recall_{probe} = \frac{X_{11}}{\sum x_{1i}} \tag{2.23}$$

$$recall_{dos} = \frac{X_{22}}{\sum x_{2i}} \tag{2.24}$$

$$recall_{u2r} = \frac{X_{33}}{\sum x_{3i}} \tag{2.25}$$

$$recall_{r2l} = \frac{X_{44}}{\sum x_{4i}} \tag{2.26}$$

Information gain: Information gain formula is not used as a final metric in this study. It is used in model selection which will be described in section 5. the Information gain is calculated using Equation (2.29) (Zhou, 2012) by the difference between two entropies. In our case first entropy is always 1, because 1 is the default value in initial state. Equation (2.27) shows how entropy is calculated for a binary class, and in binary class problem a class has value 0 or 1; as an example; in this

formula, $p(y=0)$ denotes the probability of the classes where the value equals to 0. Equation (2.27) can be generalized as seen in Equation (2.28), and $p(y_i)$ denotes the probability of each class. Equation (2.29) shows the final calculation of information gain for our study and the main idea of this formula is to find the information gain after predictions. In this formula $y_{predictions}$ denotes all unique labels in predicted data, y_p denotes each unique value in $y_{predictions}$, y denotes all original labels and $y \vee y_p$ denotes the value of y where the predicted label equals to y_p. Information gain is a well-known technique used in algorithms such as decision tree and the goal of using information gain in this algorithm is to find which feature contains more information about the result.

$$Entropy = -p(c=0) * \log_2(p(c=0)) - p(c=1) * \log_2(p(c=1)) \tag{2.27}$$

$$E(Y) = -\sum_{i=1}^{n} p(y_i) \log_2 p(y_i) \tag{2.28}$$

$$InformationGain = 1 - \sum_{y_p \in vals(y_{predictions})} p(y_p) Entropy(y \vee y_p) \tag{2.29}$$

CHAPTER THREE
RELATED WORKS: MACHINE LEARNING TO DETECT NETWORK INTRUSION

This chapter covers a broad review of machine learning (ML) applications used to detect intrusion detection. Although we only discussed three ML algorithms in section 2; in this section, we will discuss the papers that use logistic regression, naïve bayes, bayesian networks, decision trees, artificial neural networks and support vector machines to provide a broader perspective of how ML is used in network intrusion detection.

We also need to mention that although each sub-section is titled by algorithm names, those papers don't use single algorithm, but we decided to use those titles to organize this section of the thesis.

3.1 Logistic Regression

Logistic regression is an algorithm used for classification which is based on linear regression and the details of the algorithm was discussed in previous chapter.

Peng, Kou, Wang, & Shi (2011) proposes a method to solve the multi-class classification problem. In multi-class classification problems, the goal is using a binary classification algorithm as a multi-class classification algorithm. Peng (2011) uses logistic algorithm as a binary classification algorithm and applies it on KDD'99 dataset.

Gupta & Kulariya (2016) proses a method that focuses on analyzing big data. In this study, they use two different methods for feature selection, which are correlation based feature selection and chi-squared feature selection, and uses various classification algorithms including logistic regression on Spark to classify the attacks.

3.2 Bayesian Reasoning

Bayesian reasoning is an umbrella term that consists of a range of methods which uses Bayes theorem. Mitchell (1997) defines the bayesian reasoning as a method which probabilistic approach to inference.

3.2.1 Naïve Bayes

Bayes' theorem is used to find the probability of an event where conditions related to that event are used as the input. Naïve bayes (NB) is a classification algorithm that is based on Bayes' theorem with independence assumptions between features of the instances.

Amor, Benferhat, & Elouedi (2004) comparing compared the performance of naïve bayes and decision tree on the KDD Cup'99 dataset. Although decision tree obtained a higher accuracy, Naïve bayes provided better detection rates on the minor classes which are probe, u2r and r2l. Panda & Patra (2007) compared naïve bayes with artificial neural network and found that ANN is biased in favor of major classes.

Gharibian & Ghorbani (2007) compared four different machine learning algorithms (naïve bayes and gaussian classifier, decision tree and random forest). The conclusion of this study was: gaussian classifier and naïve bayes performed better on the u2r and r2l labeled attacks which are only the minor part of KDD Cup'99 dataset.

Bosin, Dessì, & Pes (2005) compared the naïve bayes with an adaptive bayesian network on KDD'99 dataset. Although they used another subset of the KDD'99 dataset, they found that the behavior of NB was the same and NB provided higher detection rates for u2r and r2l labeled attacks which are minor classes in this dataset.

3.2.2 Bayesian Networks

There are studies that used bayesian networks as a part of decision process in hybrid systems (Kruegel, Mutz, Robertson, & Valeur, 2003; Mutz, Valeur, Vigna, & Kruegel, 2006; Scott, 2004). Kruegel (2003) argues most hybrid systems result with high false alarm rates. Kruegel (2003) proposes hybrid anomaly detection system for host based intrusion detection systems and this study deploys a bayesian network to decide the final output. That proposed system was validated on the DARPA'99 dataset by Lippmann, Haines, Fried, Korba, & Das (2000), and they also compared this method to a threshold based approach. They used the same input data for both of the approaches and both of them had provided 100% true positive rates. But the false positive rate for threshold based approach was higher than the bayesian approach.

Mutz (2006) proposed a system on top of Kruegel (2003)'s study and he proposed an application based IDS which also analyzes the system calls and this study obtained improved detection rates.

3.3 Decision Trees

Decision tree (DT) algorithm is a very popular algorithm and it is used in various areas as well as in network intrusion detection. Decision trees has some benefits compared to other machine learning algorithms; they are trained quickly compared and they are interpretable. (Bouzida & Cuppens, 2006b) highlights that the rules can be extracted from decision trees to be used in expert systems. Although they have some advantages, they also have some drawbacks. Chawla (2003) and Gharibian & Ghorbani (2007) highlight that decision trees are very sensitive to the training data.

A good example of decision tree usage can be seen in the results of KDD Cup'99 where a C5.0 decision tree algorithm was used by the winner of KDD Cup'99 competition Pfahringer (2000) decision tree with ensemble methods. Gyanchandani, Yadav, & Rana (2010) also used the C4.5 decision tree algorithm in ensemble methods to improve the results of C4.5.

As we mentioned, DTs are very popular not just in security world, but also in many other areas and this help us to find lots of successful applications of DT in network intrusion like (Bouzida & Cuppens, 2006a; Chebrolu, Abraham, & Thomas, 2005; Sabhnani & Serpen, 2003) and there are various other studies which use DT as a part of a hybrid system (Depren, Topallar, Anarim, & Ciliz, 2005; Pan, Chen, Hu, & Zhang, 2003; Peddabachigari, Abraham, Grosan, & Thomas, 2007; Sinclair, Pierce, & Matzner, 1999).

Sabhnani & Serpen (2003) analyzed the performance of several machine learning techniques on the KDD Cup'99 dataset, including decision tree algorithm. Although decision trees obtained good accuracy, it did not provide good results on some classes; specifically decision trees did not perform well to detect u2r and r2l labeled attacks. Gharibian & Ghorbani (2007) found the similar results.

Although DT may show unstable results, this disadvantage is converted to an advantage by using ensemble methods by various studies. This weakness of DTs is used to build an ensemble and the very well known practical application of ensemble DTs is random forest (RF) algorithm, which is explained in section 4.6. Zhang & Zulkernine (2006) uses RF to classify the network traffic.

3.4 Artificial Neural Networks

In this section we will use artificial neural networks (ANN) as an umbrella term which covers multi layer perceptrons (MLPs) (Haykin & Network, 2004) and Self Organizing Maps (SOMs) (Kohonen, 2012).

ANNs have several advantages for the problems that require machine learning to solve. ANNs can learn complex patterns and can identify new patterns by generalizing the knowledge (Haykin & Network, 2004), which makes it a suitable solution for network intrusion. Another benefit of ANN is the flexibility to be able to work with noisy/missing data (Cannady, 1998). In their study, Debar & Dorizzi (1992) also states that ANNs are not sensitive as other machine learning algorithms

to the input data. However, ANNs have disadvantages too like other machine learning algorithms. Cannady (1998) states that: (1) a large amount of the training data is required (2) they are black boxes. Another drawback of ANN is to design the topology of an ANN which makes the training process time consuming. But, there are studies that use genetic algorithm to design the topology of an ANN (de Lacerda, De Carvalho, & Ludermir, 2000; Debar & Dorizzi, 1992; Öztürk, 2003; Yao, 1993).

3.4.1 Multi Layer Perceptrons

Multi layer perceptron (MLP) is a classification method that can be categorized under ANN term. As we mentioned in section 3.4, the topology of ANNs are very important for the success and the same rule applies for MLPs.

The studies by Bouzida & Cuppens (2006b) and Sabhnani & Serpen (2003) showed that MLP provided similar results with DT. Sabhnani & Serpen (2003) showed that the MLP provides the performance best on probe labeled attacks when it is compared to 8 other machine learning algorithms.

MLP outputs a probability for the final result and if the probability is more than 0.5, than the final results is assumed positive. If it is a multi-class classification problem, the class which has the biggest probability becomes the final result. (Bouzida & Cuppens, 2006a, 2006b) used a threshold parameter to define the final result; if the probability is higher than this threshold than the corresponding class becomes the final results otherwise it is labeled as ‘new’ class. Although the overall accuracy increased in this study, the number of ‘new’ classes increased too and this did not affect the results of u2r and r2l labeled attacks.

3.4.2 Self Organizing Maps

Self organizing maps (SOM) are not supervised machine learning algorithms, they are unsupervised which means they do not need labeled classes to be trained. SOM fall under the clustering methods which is a sub-branch of unsupervised machine learning methods.

In a study by Höglund & Hätönen (1998), they implemented a visualization tool which shows the results of SOM clusters and those clusters are based on user behaviors. The SOM is trained with data which contains normal user behavior, and SOM creates clusters that contain common behavior of the user. If there is a deviation from the clusters, this shows the possible intrusions. This study suggest that if an attack is suspected, a detailed analysis should be done to see the intrusion clearly.

Depren (2005) applied SOM as a part of a hybrid system which also contains a DT and a rule based system (RBS). In this study DT classifies the traffic RBS gives the final results based on the rules. The system is tested on KDD'99 dataset and only normal traffic is extracted to be used for SOMs. Three SOMs are trained for TCP, UDP and ICMP traffic. Depren (2005) suggests that if SOMs and DT classify the traffic as attack, the traffic is labeled as attack by the RBS. If only SOMs detect classifies the traffic as attack, the RBS can still define the output as an attack, but labels it as unknown attack type.

Differing from other studies, Thames, Abler, & Saad (2006) used normal and attack data to be used with SOM. SOM finds the clusters and then this information is passed to a NB classification algorithm so NB also learns how SOM clusters the data. The experiments were done KDD'99 dataset and the results showed that this method improved the accuracy by 3% compared to the method which only used NB.

3.5 Support Vector Machines

Support Vector Machines (SVMs) (Burges, 1998; Cortes & Vapnik, 1995) falls under supervised machine learning methods, like NB, DT and MLP as we discussed in previous sections.

SVMs can learn learn effectively from high dimensional data Boser, Guyon, & Vapnik (1992). And another advantage is training quickly when compared to MLP. Mukkamala (2002) did a comparison between MLP and SVM and found that SVM was trained in 17 seconds while MLP was trained in 18 minutes, and they obtained similar success rates.

Khan, Awad, & Thuraisingham (2007) proposed a hybrid method where he used a hierarchical clustering algorithm to cluster the data and then used SVM to train the data in each cluster. This also shortened the training time. Khan (2007) tested this proposed architecture on the DARPA'98 data set. This algorithm took 13 hours to be trained and a basic SVM trained the all data in 5 hours more.

li & Liu (2011) used SVM on KDD'99 with normalization pre-processing methods. They found that when the data is normalized before training a SVM shortens the time significantly. In this study the training time decreased from 165 seconds to 5 seconds when the min-max normalization is used but the overall accuracy only increased by 0.4%. Hasan, Nasser, Pal, & Ahmad (2014) compared SVM and random forest (RF) on KDD'99 dataset. SVM provided higher accuracy and lower false negative rate.

The method which is called one-class SVM was proposed by Schölkopf, Platt, Shawe-Taylor, Smola, & Williamson (2001) and this method is used for outlier detection in various areas and it was also applied to network intrusion topic as an anomaly detector. The anomaly detectors are trained for normal user behavior and then it is expected to produce an alarm when it sees a behavior which is not compatible with the normal user behavior. Wang, Wong, & Miner (2004) used one-

class SVM as an outlier detector and compared the results with a system called STIDE, which was developed at University of New Mexico. The normal behaviors were collected by using a UNIX system and system calls were stored as normal users' behaviors. The experiments showed that using STDIE kernels with one-class SVM can provide more accurate results.

3.6 Clustering

Clustering is a machine learning method which is used to group objects based on the given characteristics. In terms of network intrusion detection systems, clustering is used for anomaly detection where similar behaviors are grouped and outlier behaviors (attack behaviors) are grouped in another cluster.

There are several studies that use clustering for network based anomaly detection; e.g., (Eskin, Arnold, Prerau, Portnoy, & Stolfo, 2002; Leung & Leckie, 2005; Portnoy, Eskin, & Stolfo, 2001; Song, Ohira, Takakura, Okabe, & Kwon, 2008). These studies (Denning, 1987; Portnoy et al., 2001) make two main assumptions about the data: (1) that high percentage of the data contains information about normal user behavior (not intrusion data) and (2) that the attack data is statistically different from normal user behavior. Portnoy (2001) gives a specific number and says that that the data which contains normal user behavior should be more than than 98% of the total data.

Guan, Ghorbani, & Belacel (2003) proposes a clustering algorithm (Y-means) which is based on K-means clustering algorithm. K-means algorithm requires a parameter k which is used to define the number of clusters. The goal of Y-means is to make the clustering less sensitive the value k. Y-means algorithm require one more parameter which is used to set the threshold for labeling. The experimentation was done on KDD Cup'99 dataset. They obtained 82% detection rate and 2.5% false positives. But, Song (2008) used the same algorithm and used a different subset of KDD'99 dataset and reported 60% detection rate with 2.5% false positives.

CHAPTER FOUR
ENSEMBLE METHODS

In this chapter, we will cover the the definition of ensemble method. And, then we will cover widely known ensemble methods.

4.1 Introduction

Ensemble methods are techniques, where more than one model are combined to produce a better model. Ensemble methods usually produce more accurate results compared a single model. We also have seen that in competitions, where the winner was an ensemble method. In the popular Netflix Competition, the winner used an ensemble method to implement a recommendation system. The usage of ensemble methods can be also seen on kaggle.com. kaggle.com is a website where the machine learning competitions are placed and it can be seen that in almost every competition, the winner is an ensemble method.

Before making progress in this chapter, we want to explain some terms that we will use. Training data is the data which used to as an input for the algorithm. Model is the output of the machine learning algorithm after trained with the training data. This model is used to map the input to the output.

4.2 Model Generation

Stacking methods, generally, contains two phases:

- generating models by using the training dataset
- combining those models

While some ensemble methods specifically describes how the models are generated, some do not. For example averaging, which is one of the basic ensemble methods, do not describe how to generate models. In this section, we will cover the methods to generate models.

4.2.1 Manipulating Training Data

This is the most popular way of building the base models. It consists of manipulating the original training data by removing/adding instances from/to dataset or by modifying the weights in the dataset. Machine learning algorithm is trained with manipulated versions of the dataset. The individual models should show some instabilities to have a successful result for this method. Instability means that small changes in the dataset should lead to changes in the predictions of the trained model.

Bootstrap sampling is a popular method to manipulate the training data. Bootstrap sampling is a way to draw from the the original dataset with replacement, which means that the same data instance can be drawn multiple times. When the new dataset has the same size of the original dataset, this new dataset contains only 63% different instances on average. In this drawn dataset, the remaining data consists of repeated instances (Efron & Tibshirani, 1994). And, when we draw new datasets with bootstrap sampling, those new datasets will have share only 39% of the same data (Efron & Tibshirani, 1994). Bagging is an ensemble method that uses bootstrap sampling and described in section 4.6.

Beside drawing instances from the dataset, the other method for manipulating the training data is to assign weights to the instances. Changing the weights helps to control the influence of the instances in the dataset. This method is used in boosting method, which is described in Section 4.7.

4.2.2 Manipulating Input Features

This method consists of manipulating the features by removing or adding new features. The random subspace method is an example for generating models (Ho, 1998). In this method, machine learning algorithms are trained with same dataset but with different subset of randomly chosen features to generate models.

New features can also be generated to manipulate the input features. As an example; new features can be generated by using methods like principal component analysis (PCA), which is used to project the feature set into a transformed space. PCA is an example we discussed here but there are also other methods that can be used to manipulate the features (Cherkauer, 1996; Duin & Tax, 2000; Fern & Brodley, 2003).

4.2.3 Manipulating Target Variables

This technique is used to manipulate the target variable before generating each model. Error-correcting output coding (ECOC) is a method that we can show as an example that is used to generate models for this technique (Dietterich & Bakiri, 1995). ECOC is designed for multi-class classification problem. In multi-class classification problem, the final goal is to decide one class from k possible choices (where k>2). ECOC can also be considered as a meta method which combines different binary classifiers in order to solve a multi-class problem.

Table 4.1 Using ECOC for 3 class classification problem

Class	Models			
	f_1	f_2	f_3	f_4
c_0	1	1	0	0
c_1	1	0	1	0
c_2	0	1	1	1

Table 4.1 shows an example about how to create models for a 3 class classification problem. In this example; first model (f_1) is trained to understand if a training example is a member of c_o or c_1, second model (f_2) is trained to understand if a training example is a member of c_o or c_2, and so on. As it can be seen on the table, each model is used for binary classification problem. When individual models does the prediction, the predictions are combined by majority weighting, which is explained in Section 4.4.1.

4.2.4 Randomizing the Learning Algorithm

In this method, the algorithm's parameters, which can be randomized, is used to generate models. For example; neural networks assign random weights at the beginning and this leads create different models with the same training dataset.

The decision tree algorithm can randomized by splitting the nodes randomly instead of choosing the best possible split by given criterion. (Dietterich, 2000) showed that combining randomized decision trees can provide same or better results than bagging .

4.3 Averaging Based Ensemble Methods

Averaging is the most simple form of ensemble methods for numeric outputs, and this simplicity makes this method popular.

4.3.1 Simple Averaging

In this method, each model predicts a numeric value for the given input, and then, the final result is calculated by averaging the predictions. The formal definition of simple averaging can be explained as in Equation (4.1).

$$H(x)=\frac{1}{T}\sum_{1}^{T} h_i(x) \tag{4.1}$$

The simplicity of this method makes this method popular and it is considered as the first choice in many real applications.

4.3.2 Weighted Averaging

In this method, each model predicts a numeric value for the given input, and then, the final result is calculated by averaging each prediction with different weights implying different importance. Simple averaging gives the combined output H(x) as defined in Equation (4.2).

$$H(x)=\frac{1}{T}\sum_{1}^{T} w_i h_i(x) \tag{4.2}$$

Simple averaging can be considered as a special case of weighted averaging, and one should not think that weighted averaging method provides the better results than simple averaging. Reported experiments in the literature don't show that weighted averaging performs better than simple averaging (Ho, Hull, & Srihari, 1994; Kittler, Hatef, Duin, & Matas, 1998; Xu, Krzyzak, & Suen, 1992). One important reason for this conclusion is that the data in real world examples are noisy and insufficient, and that causes estimated weights to cause wrong results.

4.4 Voting Based Ensemble Methods

Voting is the simplest method for categorical values, and that makes this method to be popular. Suppose we are given a set of n individual classifiers $\{h_1, h_2, \ldots, h_n\}$ and the goal is to combine all classifiers' predictions, where the possible predictions are a member of a set of class labels $\{c_1, c_2, \ldots\}$

4.4.1 Majority Voting

In this method, every model votes for one class, and the output class of this method is the one that collects more than 50% of the votes. If none of the them receive more 50% of the votes, the combined classifier (majority voting) makes no prediction. Majority voting gives the combined output H(x) as defined in Equation (4.3).

$$H(x)=\begin{cases} c_j \; if \sum_{i=1}^{n} h_i^j(x) > \frac{1}{2}\sum_{k=1}^{l}\sum_{i=1}^{n} h_i^k(x) \\ no-prediction \quad otherwise \end{cases} \tag{4.3}$$

4.4.2 Weighted Voting

This method is also similar to majority voting but each model has different weights, which increases the importance of the models. Weighted voting gives the combined output H(x) as defined in Equation (4.4) where w_i denotes the weight of h_i.

$$H(x)=c_{\underset{i=1}{argmax}\sum_{i=1}^{n} w_i \;\; h_i^j(x)} \tag{4.4}$$

When weighted voting is used in implementations, weights are often normalized (often scaled between 0 and 1) and constrained by $w_i>0$.

4.4.3 Soft Voting

If the classifiers are providing the class probabilities, soft voting can be used. In this method, each classifier h_i outputs a l dimensional vector $(h_i^1(x), h_i^2(x), \ldots, h_i^l(x))$ where l denotes the number of possible classes.

If each classifier is treated equally, soft voting simply averages all the individual outputs and generates the output based on this calculation. This procedure can be show by Equation (4.5).

$$H^j(x)=\frac{1}{n}\sum_{i=1}^{n} h_i^j(x) \tag{4.5}$$

Weighted soft voting is also another option to be used to combine the results of classifiers when the classifiers provide the probabilities of classes. Weighted soft voting can be used in three different forms as described following:

- Weights are assigned to each classifier. The formal definition of this method is shown in Equation (4.6).

$$H^j(x) = \sum_{i=1}^{n} w_i h_i^j(x) \tag{4.6}$$

- Weights are assigned to each class. The formal definition of this method is show in Equation (4.7).

$$H^j(x) = \sum_{i=1}^{n} w_i^j h_i^j(x) \tag{4.7}$$

- Weights are assigned to the each example of each class, and this is done for each classifier. The formal definition of this method is show in Equation (4.8).

$$H^j(x) = \sum_{i=1}^{n} \sum_{k=1}^{m} w_{ik}^j h_i^j(x) \tag{4.8}$$

In practice, the third method defined above is not used because it involves a large number of weights.

4.5 Stacking Multiple Machine Learning Models

Stacking (Breiman, 1996; Smyth & Wolpert, 1998; Wolpert, 1992) is a general method where an algorithm is trained to combine the output of models. Stacking consists of two levels:

- In the first level machine learning algorithms are trained by using original training data set, and then a new data set is generated by using predictions of each model.

- In the second-level, the generated dataset is used as an input for a machine learning algorithm, which is used as a combiner method. The original class labels are used as the labels of the new training dataset as it was used in the first level.

Figures 4.1 and 4.2 show the training and the prediction phase of stacking ensemble method and the two layers that we mentioned previously can be seen on those figures.

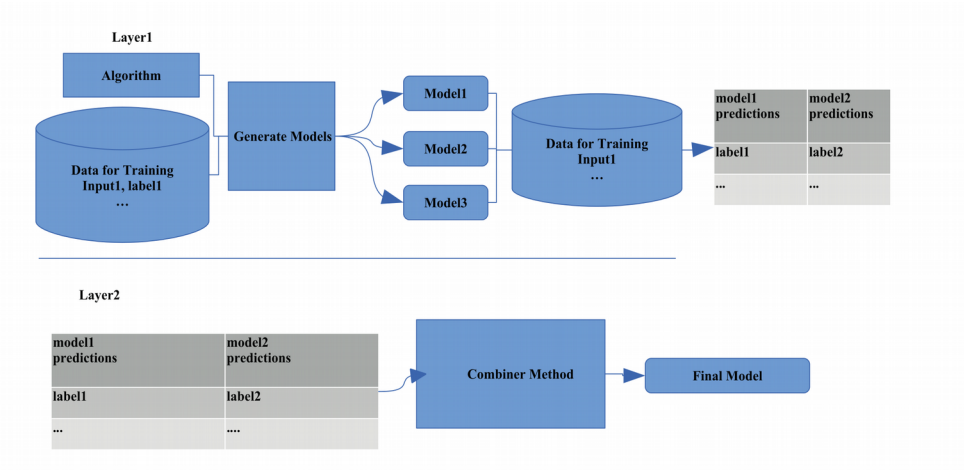

Figure 4.1 Training phase of stacking ensemble method

The first-level models can be generated by using various methods as we discussed in section 4.2 but the common way in stacking method is using different algorithms. Figure 4.3 shows the pseudo-code for stacking method which uses different algorithms to generate models. Stacking can be considered as a generic framework and it can be interpreted in two ways; 1) it can interpreted as the as a generalized version of of all ensemble methods or 2) it can be interpreted as an ensemble model where the base models are combined by using another model.

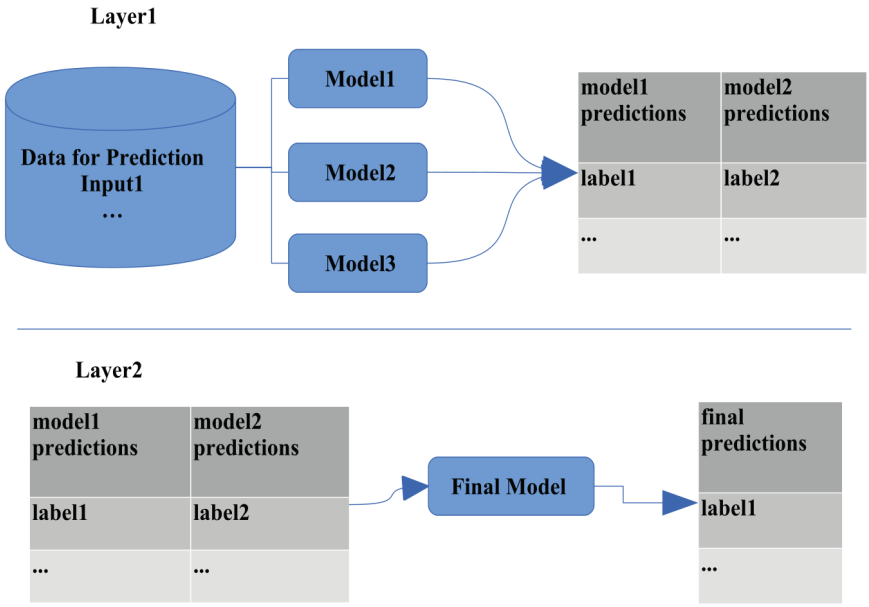

Figure 4.2 Prediction phase of stacking ensemble method

As it was mentioned, a new dataset is generated from the first-level models. If the same dataset is used again in to generate the second-level training data, over fitting may occur because because this dataset was already used in the first-level. A suggestion can be excluding the training examples which were used for generating the data set.

```
base_algorithms = [logistic_regression, decision_tree_classification, ...] #for classification

stacking_train_dataset = matrix(row_length=len(target), column_length=len(algorithms))
stacking_test_dataset = matrix(row_length=len(test), column_length=len(algorithms))

for i,base_algorithm in enumerate(base_algorithms):
    stacking_train_dataset[,i] = base_algorithm.fit(train, target).predict(train)
    stacking_test_dataset[,i] = base_algorithm.predict(test)

final_predictions = combiner_algorithm.fit(stacking_train_dataset, target).predict(stacking_test_dataset)
```

Figure 4.3 Stacking algorithm

Breiman (1996) demonstrated the success of stacking method for regression. In this study, Breiman (1996) used regression decision trees with different sizes or linear regression algorithm with different numbers of variables as the first-level. Breiman (1996) also used linear regression algorithm as the second-level algorithm

and put a constraint that coefficients of all regressions should be positive. Breiman (1996) emphasized that this constraint is an important factor for the performance of stacked ensemble when it is compared to the single best learner.

Ting & Witten (1999) recommended using the class probabilities as features for the training dataset which is used in second level, because they mention that this makes it possible to take into account both the predictions and the confidences of the classifiers.

4.6 Bootstrap Aggregating

The Bootstrap Aggregating is also know as bagging. In this method, the models are generated by training the same algorithm but with random subsets of the original training dataset. These subsets are drawn from the original training dataset with bootstrap sampling method. Bootstrap sampling is a method where the we draw examples from the dataset but do not remove the selected example from the original dataset. When this method is used, some original examples may appear more than once or some original examples may not be present in the drawn subset. As an example; if we want to draw a subsample with m elements, we need to select a random example out of the original dataset m times, without removing the selected elements from the original dataset.

The second step in this method is combining those models. Methods such as voting and averaging (section 4.3 and 4.4) are used for this purpose. In bagging, each subset can be generated independently which helps the implementation to be done in parallel in training phase.

Random Forest is a widely known algorithm and it uses the bagging method. Figure 4.4 shows the pseudo-code for bagging.

```
def bootstrap_sample(original_dataset, m):
    sub_dataset = []
    for i in range(m):
        sub_dataset.append(
            random_one_element(original_dataset)
        )
    return sub_dataset

def bagging(n, m, base_algorithm, train_dataset, target, test_dataset):
    predictions = matrix(row_length=len(target), column_length=n)
    for i in range(n):
        sub_dataset = bootstrap_sample(train_dataset, m)
        predictions[,i] = base_algorithm.fit(original_dataset, target).predict(test_dataset)

    final_predictions = voting(predictions) # for classification
    final_predictions = averaging(predictions) # for regression

    return final_predictions
```

Figure 4.4 Bootstrap aggregating algorithm

4.7 Boosting

The term “boosting” is used to describe the machine learning algorithms which aim to evolve the weak models to stronger models. A model is called a weak model if it the error rate is substantial, but the performance is not random, which means the accuracy is more than 50% for a binary classification. Boosting is built step by step and does the training with the same dataset in each step but adjusts the weights of the examples based on the error of the last model’s prediction. The main idea of boosting is to force the models to focus on the examples in dataset which are hard to learn. Boosting runs sequentially, so it can’t be implemented to run in parallel.

Adaboost is a widely known algorithm which is considered as a boosting machine learning algorithm. The founders of Adaboost won the Gödel Prize for this algorithm. Figure 4.5 shows the pseudo-code for boosting.

```
def adjust_dataset(_train, errors):
    #create a new dataset by using the hardest instances
    ix = get_highest_errors_index(train)
    return concat(_train[ix], random_select(train))

models = []
_train = random_select(train)
for i in range(n): #n rounds
    model = base_algorithm.fit(_train)
    predictions = model.predict(_train)
    models.append(model)
    errors = calculate_error(predictions)
    _train = adjust_dataset(_train, errors)

final_predictions = combine(models, test)
```

Figure 4.5 Boosting algorithm

CHAPTER FIVE
USING STACKING ENSEMBLE ON NETWORK INTRUSION DETECTION

In this study, we applied a stacking ensemble method on network intrusion. We will cover the topics related to this study in this chapter. In the following sub-section, we will re-cap stacking method that we explained in section 4.5 and then we will explain how we used stacking ensemble method. In section 5.2. we will describe how we conducted experiments and share the results.

5.1 Our Approach: Parameterized Stacking Ensemble Method

5.1.1 Stacking

Stacking is a type of ensemble method as we discussed in section 4. In the training phase, generally the base classification algorithm(s) and training data are accepted as inputs. Model generation is used to train the algorithm with data and generate models. If the stacking implementation accepts only one algorithm, usually the "model generation" phase generates n models by using the algorithm with randomly drawn sub-datasets. If the implementation accepts multiple algorithms, usually the training set is trained with each algorithm. When the models are generated, these models generate the predicted labels. This is the first layer of the two-layered training phase. The predicted labels of each model are given as input to the second layer. In the second layer, a classification algorithm (also called the combiner method) is used to generate a final model while the original labels are still used for labeling the new training data. Figure 4.1 shows the training phase of the stacking approach.

The prediction phase of the stacking also contains two layers. The first layer uses the input data and previously generated models and makes a prediction, and the second layer uses the model previously generated in the second layer of the training phase. Figure 4.2 shows the prediction phase of stacking approach.

5.1.2 Model Generation

In this study, we used logistic regression and the training dataset as inputs to the first layer. The modified 'model generation' consists of two steps:

- Random feature selection and training,
- Selecting the best 10 models.

In the first step, random sub-datasets were not drawn by rows but by features. In the original paper on random forests (Breiman, 2001), which is an ensemble algorithm, the author emphasized that a random forest creates the best results when it is used with random feature selection. There are also various papers that investigated the power of random feature selection in ensemble methods. Bryll, Gutierrez-Osuna, & Quek (2003) showed that using random feature selection for ensemble classifiers over-performs and mentioned that using random feature selection helps to create non-correlated models, and those non-correlated models help to improve the performance of the ensemble method.

Skurichina & Duin (2005) mentioned that while combining multiple feature subsets, random feature selection may be preferred as it may provide more successful results in obtaining independent feature subsets. In our study, we selected n/2 features, where n equals the number of features, in each draw and generated 100 sub-datasets, which led to 100 models.

The second step reduces 100 models to 10 models. The best 10 models are selected according to the three evaluation methods of accuracy, information gain, and recall; the higher the value, the better the model is. The formula of each metric was discussed in section 2.3. This process is shown in Figure 5.1.

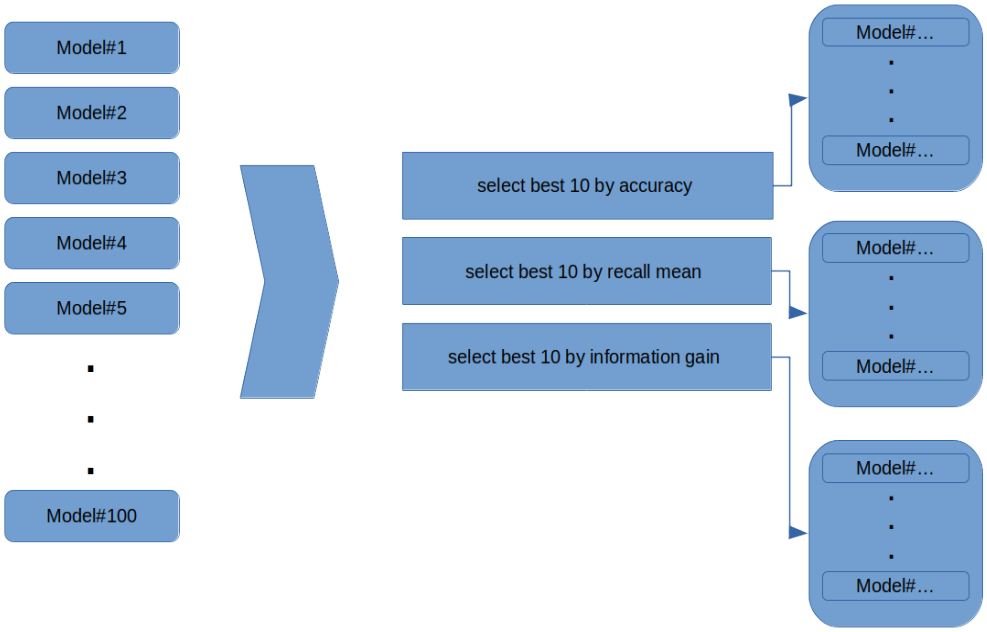

Figure 5.1 Model selection for parametrized stacking ensemble method

Using 100 and 10 as the parameters is a common practice in ensemble implementations. For example, the random forest's default 'number of estimators' parameter is 10 and xgboost's default 'number of estimators' is 100 in a widely used Python library called scikit-learn (Pedregosa et al., 2011).

A confusion matrix is used to show the performance of the model by providing how it performs per label.

Evaluation metrics to select models are as follows:

- Accuracy: While using accuracy as an evaluation method for selecting the 10 best models, each model's accuracy is calculated and then the 10 highest scores are selected for the next step. The details of the accuracy formula was discussed in section 2.3.

- Information gain: As in accuracy calculation, information gain of each model is calculated and 10 best scores are chosen. The details of information gain

was discussed in section 2.3. In our study, we used information gain to select best models, which affects the results.

- Recall (true positive rate) mean: Recall mean also known as true positive rate mean is calculated as the mean of the recall value for each label (Probe, DoS, U2R, R2L). We discussed the recall and recall mean formulas in section 2.3. Another option could be precision, but recall helps us to understand if the model generalization is working or not; while precision allows us to understand if an attack is a real attack. For example; if a model detects only one attack and it is a real attack, the precision will be 100% but the generalization will be very poor.

5.2 Experiments

In this study, we used stacking ensemble approach by using different base classification algorithms as stacking algorithm. In this section, we will describe the steps of the experiments. Finally, we will describe evaluation metrics.

5.2.1 The Steps of The Experiments

In this study, we conducted 13 different experiments by using 4 main methods. In the first method, we did not use any stacking and used logistic regression for classification of data, which is compared against other methods. The other three methods share three common steps, but in the last step, each method uses a different algorithm as a combiner method. The following steps are the shared common steps among the three methods:

- using logistic regression to generate models,
- selecting models by using 4 different methods,
- using a combiner method.

Model selection methods, which are used in the second step, and the algorithms used by each method have been discussed in section 5.1.

In our study, we did not create a new set of models in each experiment and we used all 100 models in all experiments. In other words, we created 100 models once and used them again and again in each experiment. 100 models were reduced to 10 models in each experiment by using one selection method at a time. This gave us the chance to understand how the combiner and selection methods perform.

We conducted 13 experiments. In the first experiment, we used logistic regression without stacking. For the rest of the experiments, we used three different combination algorithms and four different model selection methods for each combination algorithm, which results in 12 experiments. The details of each experiment can be found in Table 5.1 where the results are reported.

We implemented this experiment with python programming language and used scikit-learn machine learning library (Pedregosa et al., 2011) on Ubuntu Linux OS. Scikit-machine learning library is a widely used library for academic and commercial purposes. It contains implementations for both supervised learning (classification and regression) and unsupervised learning (clustering) algorithms. It allows developers to use a common interface for all algorithms, which makes it easier to use and develop prototypes fast. Appendix-2 contains the detailed description of each library we used.

5.2.2 The Results of The Experiments

We evaluated each experiment with six metrics, which sum up to 78 results. The evaluation metrics are listed below:

- tp_normal: True positive percentage of normal labeled data,
- tp_probe: True positive percentage of probe labeled data,
- tp_dos: True positive percentage of DoS labeled data,
- tp_u2r: True positive percentage of u2r labeled data,

- tp_r2l: True positive percentage of r2l labeled data,
- accuracy: Accuracy of the experiment.

The formula of the metrics were described in section 2.3

Table 5.1 Results of the experiments (true positive rates)

Combiner/ Stacking Algorithm	**Model Selection Method**	**Exp. #**	**tp normal %**	**tp probe %**	**tp dos %**	**tp u2r %**	**tp r2l %**
Logistic Regression (no stacking)	-	1	98.1	68.7	82.8	8.6	5.6
Logistic Regression	all	2	98.3	**83.5**	**97.3**	12.9	5.2
	accuracy score	3	98.4	72.9	97.3	8.6	5.6
	information gain	4	98.4	73.7	97.3	7.1	5.6
	recall mean	5	98.2	72.3	97.3	8.6	5.6
Decision Tree	all	6	**99.4**	77.0	97.3	14.3	2.5
	accuracy score	7	99.0	73.7	97.3	7.1	5.6
	information gain	8	99.0	78.3	97.3	8.6	5.7
	recall mean	9	99.0	75.2	97.3	18.6	5.5
Naïve Bayes	all	10	97.3	78.9	93.2	**47.1**	6.0
	accuracy score	11	97.9	80.2	97.2	30.0	5.7
	information gain	12	97.9	79.9	97.2	30.0	**7.6**
	recall mean	13	98.0	79.0	96.7	35.7	5.6

We provided the results in Table 5.1 and Table 5.2 and compared the results with other studies in Table 5.3 and provided the datasets used in those studies in Table 5.4. True positive rates are provided in Table 5.1 and accuracy scores are given in Table 5.2. We also provided four bar plots (Figures 5.2-5.5) to show the results of trues positive rates of attacks (probe, dos, u2r and r2l) in experiments. Appendix-3 contains the confusion matrices of all 13 experiments and the percentages shown in Table 5.1 are based on the tables listed in Appendix-3.

Table 5.2 Results of the experiments (accuracy)

Combiner/ Stacking Algorithm	**Model Selection Method**	**Exp. #**	**Accuracy %**
Logistic Regression (no stacking)	-	1	81.53
Logistic Regression	all	2	92.43
	accuracy score	3	92.33
	information gain	4	92.36
	recall mean	5	92.31
Decision Tree	all	6	92.47
	accuracy score	7	92.46
	information gain	8	**92.55**
	recall mean	9	92.46
Naïve Bayes	all	10	89.19
	accuracy score	11	92.29
	information gain	12	92.39
	recall mean	13	91.90

We will discuss our comments under 4 different categories to make it easy to follow:

- Using logistic regression without stacking:
  - Experiment #1 uses logistic regression and does not use stacking method. We decided to use this experiment as a base benchmark. This experiment resulted with the following true positive rates 98.1%, 68.7%, 82.8%, 8.6%, 6.5% and 81.53% for

normal, probe, dos, u2r, r2l and normal labels, respectively. Accuracy of this experiment provides the worst performance. We also need to mention that true positive rates for dos and probe are the worst results among all experiments. Figures 5.2 and 5.3 plot the true positive rate of probe and dos labeled traffic, where it also can be seen that experiment #1 provides the worst results for those labels.

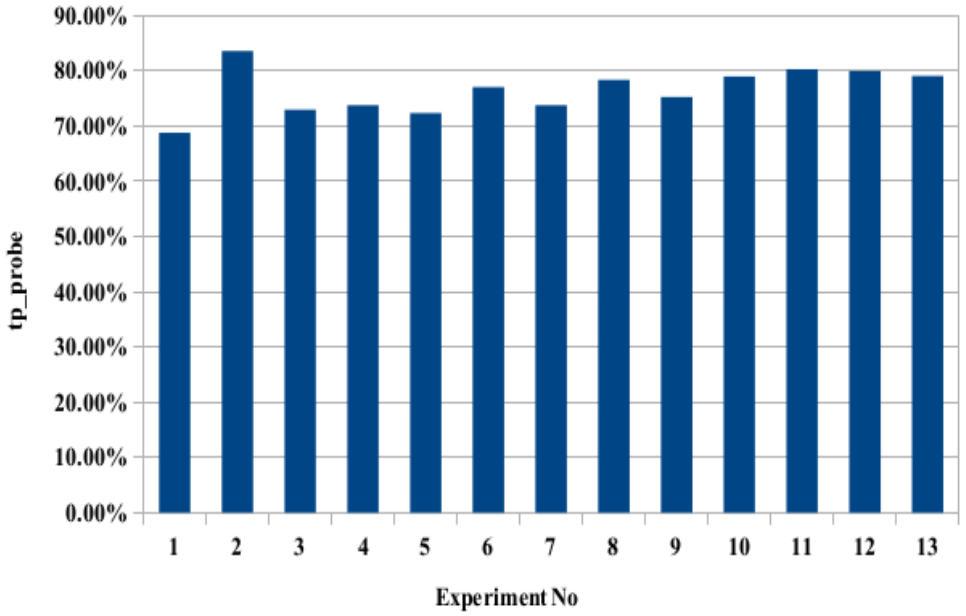

Figure 5.2 Bar plot of tp_probe values of experiments' results

- Using logistic regression as combiner method:

  - Experiment #2 uses logistic regression as a combiner method and uses all the models generated in the first layer of stacking method. This experiment resulted with the following true positive rates 98.3%, 83.5%, 97.3%, 12.9%, 5.2% and 92.43% for normal, probe, dos, u2r, r2l and normal labels, respectively. This experiment provides the highest true positive rates for dos and probe attacks. Using all base models as input for logistic regression combiner method provides a good diversity for probe labeled attacks so that stacking can handle the diversity among all base classifiers. This results also can be seen on Figures 5.2 and 5.3 .

◦ Experiment #3 uses logistic regression as a combiner method and filters all models by using the accuracy score of each model. This experiment resulted with the following true positive rates 98.4%, 72.9%, 97.3%, 8.6%, 5.6% and 92.33% for normal, probe, dos, u2r, r2l and normal labels, respectively. Experiment #4 uses logistic regression as a combiner method and filters all models by using the information gain of each model. This experiment resulted with the following true positive rates 98.4%, 73.7%, 97.3%, 7.1%, 5.6% and 92.36% for normal, probe, dos, u2r, r2l and normal labels, respectively. Experiment #5 uses logistic regression as a combiner method and filters all models by using the recall mean metric of each model. This experiment resulted with the following true positive rates 98.2%, 72.3%, 97.3%, 8.6%, 5.6% and 92.31% for normal, probe, dos, u2r, r2l and normal labels, respectively.

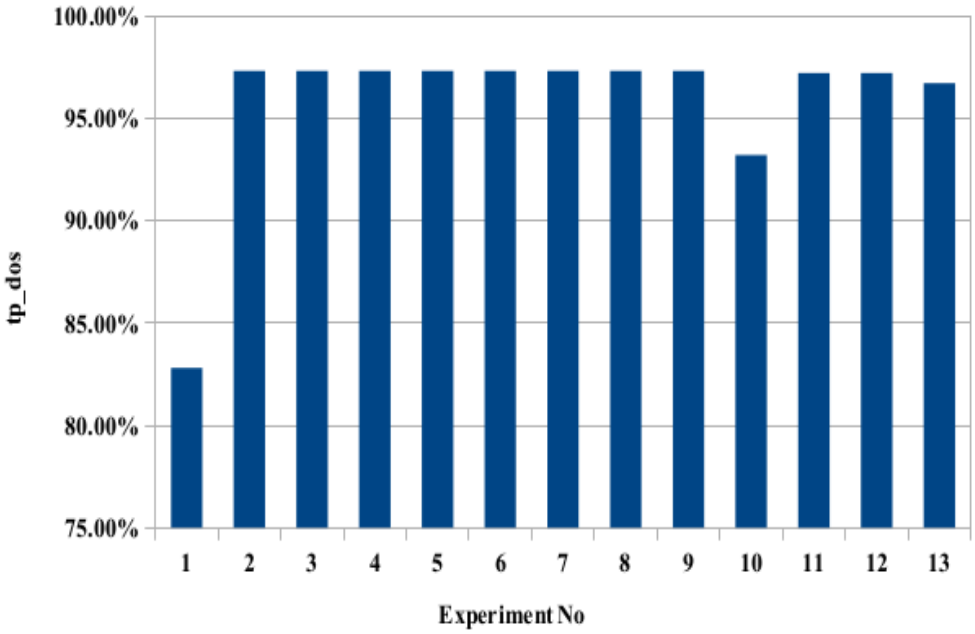

Figure 5.3 Bar plot of tp_dos values of experiments' results

◦ Those three experiments (experiment #3, #4 and #5) use a model selection process before using the logistic regression as combiner. We can see that true positive rate for probe drops to 72.9%, 73.7% and 72.3% from 83.5% compared to experiment #2. The same case happens for true positive rates for u2r and accuracy; u2r true positive rate drops to 8.6%,7.1 and 8.6% from 12.9%., accuracy drops from

92.43% to 92.33%, 92.36% and 92.31%. Even though, those metrics drops, we can see that r2l true positive rate increases from 5.2% to 5.6% for all experiment #3, #4 and #5. This shows that using those selection methods (accuracy_score, information_gain and recall_mean) reduces the diversity of classifiers and generalization under-performs but helps to increase the true positive rate for r2l which is a minor label.

- Using decision tree as combiner method:

  - Experiment #6 uses decision tree as a combiner method and uses all the models generated in the first layer of stacking method. This experiment resulted with the following true positive rates 99.4%, 77.0%, 97.3%, 14.3%, 2.5% and 92.47% for normal, probe, dos, u2r, r2l and normal labels, respectively. This experiment provides the highest true positive rate for normal labeled traffic. Using all base models as input for decision tree combiner method provides a good diversity for normal labeled traffic so that stacking can handle the diversity among all base classifiers in favor of normal labeled traffic.

  - Experiment #6 provides an outlier for r2l true positive rate, which is 2.5%. Decision tree algorithm tends to over-fit data when the depth of the tree increases. We are using all 100 models for experiment #6 and that is causing the tree to create more sub-trees and increase the depth compared to experiments #7-9. Since r2l is a minor label, the decision tree tends to over fit, r2l true positive rate takes the disadvantage of this situation.

  - Experiment #7 uses decision tree as a combiner method and filters all models by using the accuracy score of each model. This experiment resulted with the following true positive rates 99.0%, 73.7%, 97.3%, 7.1%, 5.6% and 92.46% for normal, probe, dos, u2r, r2l and normal labels, respectively. Experiment #8 uses decision tree as a combiner method and filters all models by using the information gain of each model. This experiment resulted with the following true positive rates 99.0%, 78.3%, 97.3%, 8.6%, 5.7% and 92.55% for normal, probe, dos, u2r, r2l and

normal labels, respectively. Experiment #9 uses decision tree as a combiner method and filters all models by using the recall mean metric of each model. This experiment resulted with the following true positive rates 99.0%, 75.2%, 97.3%, 18.6%, 5.5% and 92.46% for normal, probe, dos, u2r, r2l and normal labels, respectively.

- We see a similar pattern between using logistic regression as combiner method (experiment #2-5) and using decision tree as combiner method (experiment #6-#8) for r2l. The value of this metric is higher when we don't use a model selection method.

- Experiment #9 provides the highest value for u2r true positive rate among the experiments #6-#9. Experiment #9 uses recall mean as selection method where the bases models are filtered by the mean of all true positive rates.

- Even though experiment#8 do not provide the best results for true positive rates, it has the highest accuracy value. The main reason for this result is; this experiment made true predictions for major labels more than other experiments. Accuracy calculation score is calculated based on the true correct predictions without taking into account the true positive rates for each label.

• Using naïve bayes as combiner method:

- Experiment #10 uses naïve bayes as a combiner method and uses all the models generated in the first layer of stacking method. This experiment resulted with the following true positive rates 97.3%, 78.9%, 93.2%, 47.1%, 6.0% and 89.19% for normal, probe, dos, u2r, r2l and normal labels, respectively. This experiment provides the highest score for u2r true positive rate. In section 3, we have discussed that bayesian methods provide the good results for minor labels. This shows us that we have transferred the information correctly from stacking method's level-1 to level-2. This result also can be seen on Figure 5.4, which plots the true positive rates of u2r for all experiments.

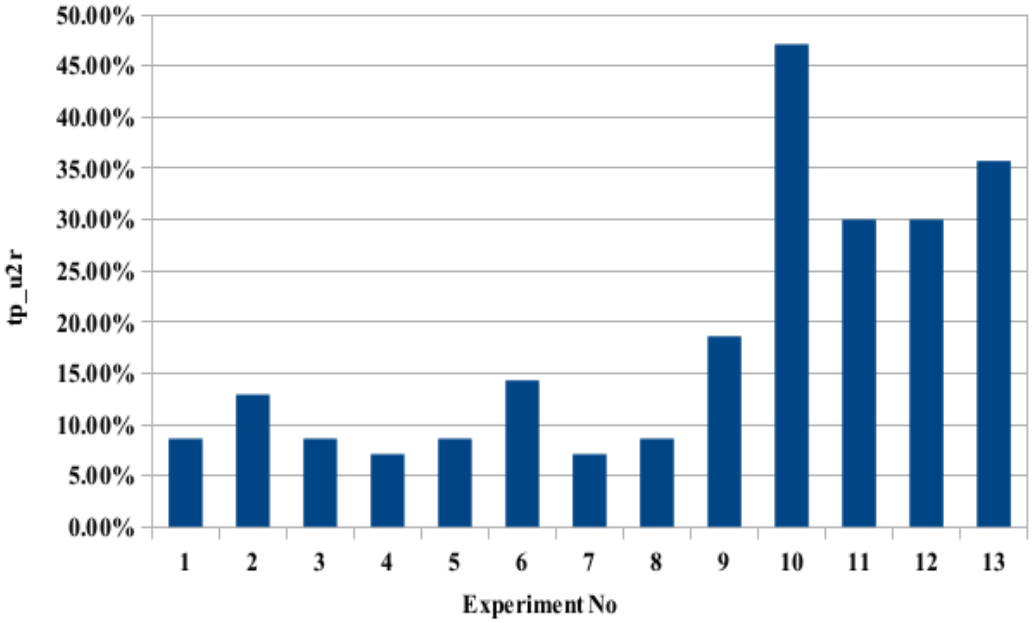

Figure 5.4 Bar plot of tp_u2r values of experiments' results

◦ Experiment #11 uses naïve bayes as a combiner method and filters all models by using the accuracy score of each model. This experiment resulted with the following true positive rates 97.9%, 80.2%, 97.2%, 30.0%, 5.7% and 92.29% for normal, probe, dos, u2r, r2l and normal labels, respectively. Experiment #12 uses naïve bayes as a combiner method and filters all models by using the information gain of each model. This experiment resulted with the following true positive rates 97.9%, 79.9%, 97.2%, 30.0%, 7.6% and 92.39% for normal, probe, dos, u2r, r2l and normal labels, respectively.

◦ Experiment #13 uses naïve bayes as a combiner method and filters all models by using the recall mean metric of each model. This experiment resulted with the following true positive rates 98.0%, 79.0%, 96.7%, 35.7%, 5.6% and 91.90% for normal, probe, dos, u2r, r2l and normal labels, respectively.

◦ Experiment #12 provides the highest value for r2l true positive rate among all 13 experiments. This experiment uses information gain for model selection. We can similar patterns among all experiments where information gain works in favor of u2r

true positive rate. Figure 5.5 shows the r2l true positive among all experiments and this result can be seen on that figure too.

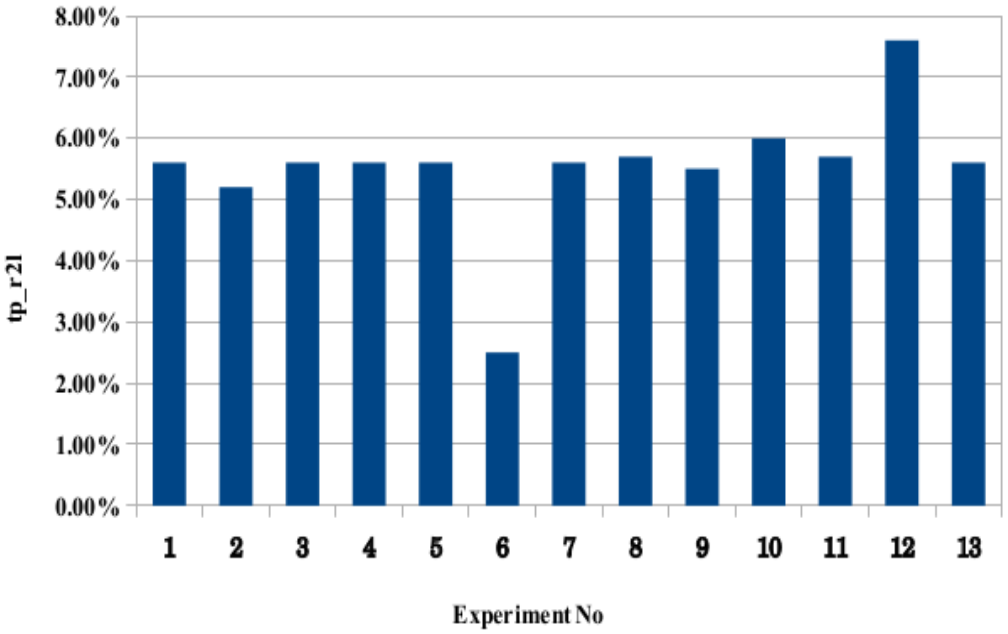

Figure 5.5 Bar plot of tp_r2l values of experiments' results

- We can see the using naïve bayes as combiner method provides the best results for u2r and r2l true positive rates, and those labels are minor labels. An algorithm that less tends to over-fit may provide better results for an unbalanced dataset. Naïve bayes tends to over fit less than other classification algorithms because it does not converge at all, it learns its parameters by calculating.

In Table 5.3, the results of parameterized stacking ensemble method are compared to other studies and this table also shows which study uses which component of KDD'99 dataset for training and testing purposes. In this table, we used the studies, which provide detection rates for each label, for comparison. Toosi & Kahani (2007) constructed a custom ensemble that uses genetic algorithm and fuzzy algorithm. Compared to this study, our ensemble technique that combines all generated models with Naïve Bayes, provides better detection rate for u2r, but not for others. Agarwal & Joshi, (2001) introduced a rule based framework, Gunes Kayacik, Nur Zincir-

Heywood, & Heywood (2007) introduced a self-organizing map (SOM) based technique and Pfahringer (2000) explained the technique, which is the winner of KDD99, that uses bagged boosting ensemble technique. Our study provides better accuracy results for all types of attacks compared to Agarwal & Joshi (2001), Gunes Kayacik (2007) and Pfahringer (2000). Al-Yaseen, Othman, & Nazri (2015) used a modified k-means technique and our study provides better detection rates for u2r and probe compared to Al-Yaseen (2015). Bouzida & Cuppens (2006b) compared decision tree and backpropagation neural network and the results of applying neural network can be seen in Table 5.3, neural network did not detect any of u2r labeled attacks because of over-fitting problem. Parameterized stacking ensemble method and other studies never provided a result with 0% in any of the categories.

Table 5.3 Comparison of the results

Study	**Normal** (%)	**Probe** (%)	**DoS** (%)	**U2R** (%)	**R2L** (%)	**Accuracy** (%)
Toosi & Kahani (2007)	98.2	84.1	99.5	14.1	31.5	N/A
Dietterich (1998)	99.5	73.2	96.9	6.6	10.7	92.59
Agarwal & Joshi (2001)	92.4	72.8	96.5	22.9	11.3	N/A
Kayacik (2007)	99.5	83.3	97.1	13.2	8.4	93.3
Pfahringer (2000)	98.55	80.9	99.6	16.2	31.6	95.71
Bouzida & Cuppens (2006b)	97.87	71.63	97.0	0.0	26.68	N/A
Parameterized Stacking Ensemble Method	*99.4*	*83.5*	*97.3*	*47.1*	*7.6*	*92.55*

Table 5.4 Comparison of the datasets

Study	Training Data	Testing Data
(Toosi & Kahani (2007)	10% of KDD	Corrected KDD
Dietterich (1998)	Whole KDD	Corrected KDD
Agarwal & Joshi (2001)	10% of KDD	Corrected KDD
Kayacik (2007)	Whole KDD	Corrected KDD
Pfahringer (2000)	10% of KDD	Corrected KDD
Bouzida & Cuppens (2006b)	10% of KDD	Corrected KDD
Parameterized Stacking Ensemble Method	10% of KDD	Corrected KDD

5.3 Algorithmic Time and Space Complexity

In this study, we discussed how parameterized stacking ensemble method works on a dataset; but we also need to discuss if this approach can be used on real-time data. There are two steps that should be implemented to apply this study in real-time applications; 1) extract features from the raw data 2) apply the machine learning models for prediction. Assuming first step is already applied, we will focus on the second step. In this sub-section, we will not discuss the training phase of the models, but will discuss the prediction phase of the models because those parts are deployed as the second step. We will discuss the algorithmic time complexity and come up with a conclusion if it can be used in a real-time environment.

The standard order notation will be used where O(n) indicates the quantity which grows at most linearly with n, $O(n^2)$ indicates the quantity which grows at most quadratically with n, and so on.

In this study, 3 machine learning algorithms were discussed as base algorithms; logistic regression, decision tree and naive bayes. Logistic regression uses the formula that was given as Equation (2.1) in prediction phase. It contains a loop

which fetches the weights and multiplies by the given input features, the complexity of logistic regression can be shown as O(m) where m denotes the number of attributes. Witten & Frank (2016) discuss that the depth of a decision tree is O(log n) and since the decision tree's prediction phase's time complexity is based on the depth of a decision tree, a decision tree's time complexity is O(log n) where n denotes the number of examples used in the training dataset. Naïve bayes needs to fetch each attribute's likelihood probability and the prior probability of each class as we discussed in Section 2.1.3 and it ends up with the time complexity O(mc) where m denotes the number of attributes and c denotes the number of classes (Fleizach & Fukushima(1998)). Those will be combined in stacking algorithm which ends up as: O(m) + O(log n)+ O(mc)

We will compare this time complexity to the idea of what would the complexity be if it was implemented as policy based firewall. Acharya & Kumar (2016) shows that theory of such policies indicates that the size of the decision tree is $O((2n)^m)$, where the policy has n rules and examines m features of packets and time complexity for this decision tree is O(nm).

Since $m<n$ and $c<n$, $O(m) + O(\log n)+ O(mc) \subseteq O(nm)$ shows that the time complexity of parameterized stacking ensemble method is not worse than the current policy based implementations.

Logistic regression requires to fetch only the weights of the features on prediction phase and this leads to space complexity O(m) where m denotes the number of attributes. Decision tree's worst case space complexity is O(nm) where n denotes number of examples in the training dataset and m denotes the number of attributes. Naïve bayes is required to store likelihood and prior probabilities and this leads to a space complexity O(m)+O(c) where m denotes the number of attributes and c denotes the number of classes (Fleizach & & Fukushima(1998)). Stacking combines those algorithms and the space complexity becomes (k/3) * (O(m)+O(nm)+O(m) +O(c)) where k denotes the number of base models.

Since k<m, m<n and c<n, (k/3) * (O(m)+O(nm)+O(m)+O(c)) $\subseteq$ O((2n)^{m}) shows that the space complexity of parameterized stacking ensemble method is not worse than the current policy based implementations.

CHAPTER SIX
USING GENETIC ALGORITHM IN STACKING ENSEMBLE FOR BASE MODEL SELECTION

In this chapter, we will apply the stacking ensemble method on network intrusion again, but this time, we will use genetic algorithm to solve the problem of selecting the base models and we will use an oversampling method to overcome the problem of minor classes.

6.1 Our Approach: Genetic Algorithm Based Base Model Selection for Stacking Ensemble

The approach of this study is slightly different from the approach we discussed in section 5.1. We used an oversampling method to balance the data and we used genetic algorithm to find the base models instead of filtering them by using a fixed value.

Figure 6.1 shows the data preprocessing step which is used in this study. We discussed the details of one hot encoding in section 2.2.1, normalization in section 2.2.2 and over sampling in section 2.2.3.

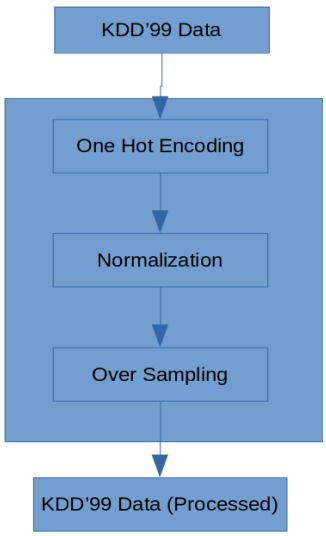

Figure 6.1 KDD'99 preprocessing

In this study we used the same technique as we used in section 5.1 for model generation, but we created 50 base models. Although we used more than one combiner method in section 5, in this study we used only naïve bayes as the combiner method. The main purpose of selecting this algorithm as combiner method is; it provided highest true positive rates for u2r and r2l labeled network traffic, which are considered minor labels. Another main reason is; it does not converge by its nature.

As we discussed, we used genetic algorithm to detect the best combination. For this purpose, we need to represent the base models in terms of genetic algorithm. Base models are used as chromosomes of each individual of the population which are used in genetic algorithm. Each individual consists of 50 chromosomes and each chromosome can take 0 or 1. 0 means that respective base model will not be used while 1 means that that respective base model will be used. Table 6.1 shows the initial population with 10 individuals, where c#1, c#2, …, c#50 represents the basemodel#1, basemodel#2, …, basemodel#10.

Table 6.1 Initial population

	c#1	**c#2**	**c#3**	**c#4**	**...**	**c#49**	**c#50**
individual#1	1	0	1	1	...	1	1
Individual#2	0	1	1	1	...	0	0
...	...	...	...	...	...	...	...
Individual#10	1	0	0	1	...	0	1

We applied the steps of the genetic algorithm as we described in section 2.1.4. The objective function returns 5 values which consist of true positive rates of each label. Since the goal is to optimize 5 different metrics at the same time, this problem becomes a multi-objective optimization problem. We discussed multi-objective optimization based genetic algorithm in section 2.1.4.

In the study we discussed in section 5, we used various metrics to select the base models and we used those models for the second layer of stacking. In this study, we used genetic algorithm to select the base models and we did not provide a specific number as the count of the base models to be selected.

6.2 The Result of The Experiment

Genetic algorithm tries to optimize the results iteratively. Each iteration contains a population where each individual in the population is a combination of base models. Figures 6.2 to 6.6 show the minimum value of true positive rate of normal, dos, probe, u2r and r2l labeled traffic in each iteration. Figure 6.7 shows the minimum overall accuracy in each iteration. The reason to show the minimum values is to express how genetic algorithm evolved by iterations.

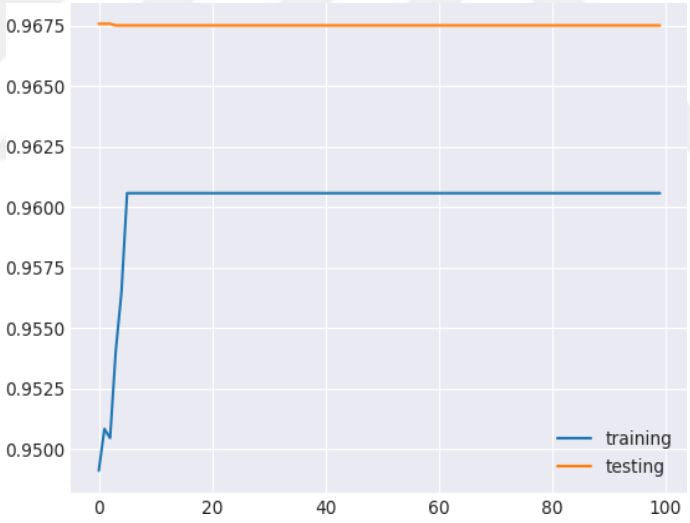

Figure 6.2 Minimum true positive rate of normal labeled traffic in each iteration

Figure 6.2 shows the minimum true positive rate of normal labeled traffic in each iteration. It can be seen that, over time, the genetic algorithm converges and it does not increase the true positive rate after fifth iteration. The true positive rate of training dataset increase from 95% to 96%, while the true positive rate of testing has a very small drop which from 96.75% to 96.70%.

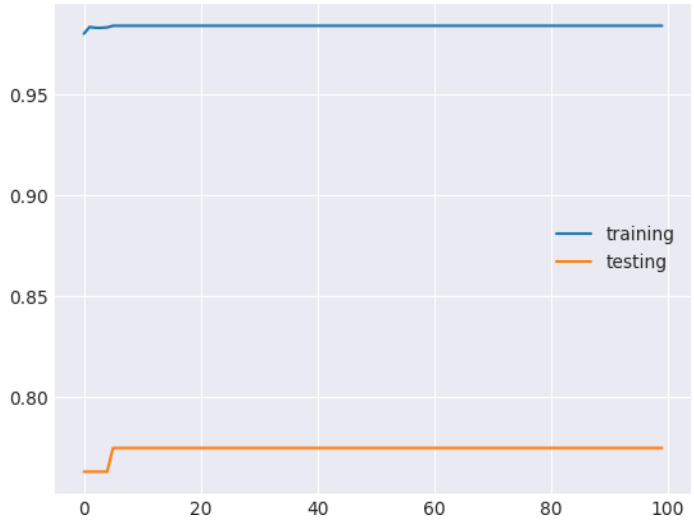

Figure 6.3 Minimum true positive rate of probe labeled traffic in each iteration

Figure 6.3 shows the minimum true positive rate of probe labeled traffic in each iteration. It can be seen that true positive rates of probe in both training and testing increases and stops increasing after the fifth iteration.

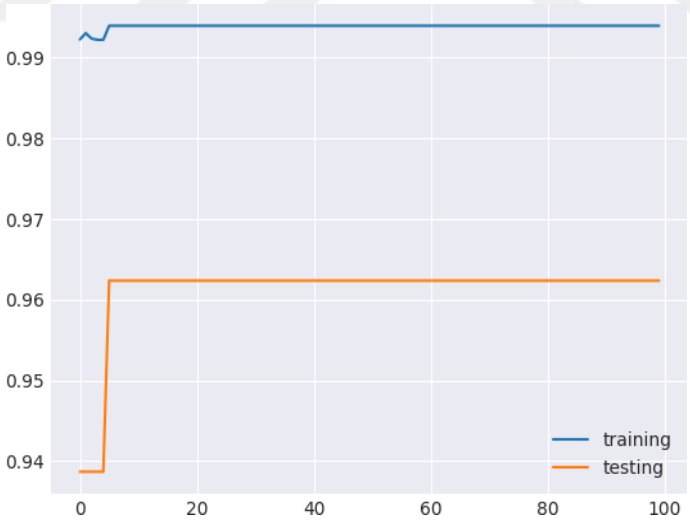

Figure 6.4 Minimum true positive rate of dos labeled traffic in each iteration

Figure 6.4 shows the minimum true positive rate of dos labeled traffic in each iteration. It can be seen that true positive rates of probe in both training and testing increases even though the true positive true rate of training dataset drops in second

and third iterations. The increase is 2% in testing dataset while the true positive rate is already over 99% in training dataset.

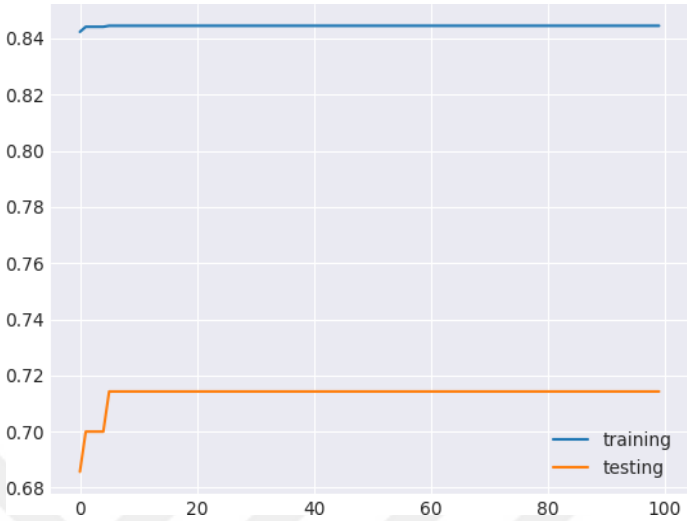

Figure 6.5 Minimum true positive rate of u2r labeled traffic in each iteration

Figure 6.5 shows the minimum true positive rate of u2r labeled traffic in each iteration. This graph shows that true positive of u2r increases both in training and testing data and it is a very small increase training dataset.

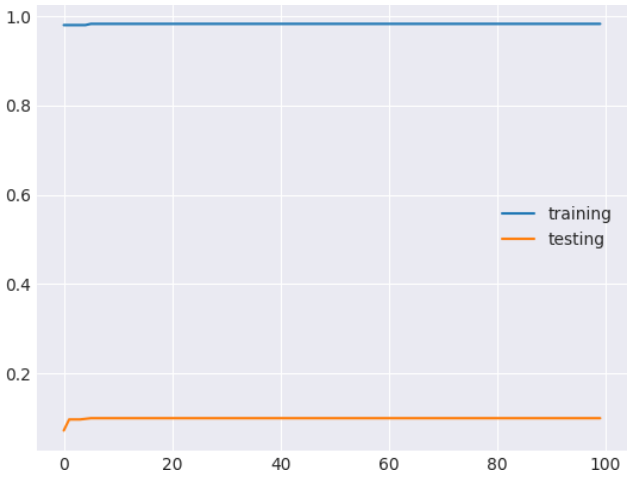

Figure 6.6 Minimum true positive rate of r2l labeled traffic in each iteration

Figure 6.6 shows the minimum true positive rate of r2l labeled traffic in each iteration. This graph shows that true positive of r2l is almost 100% in the first iterations for the training dataset.

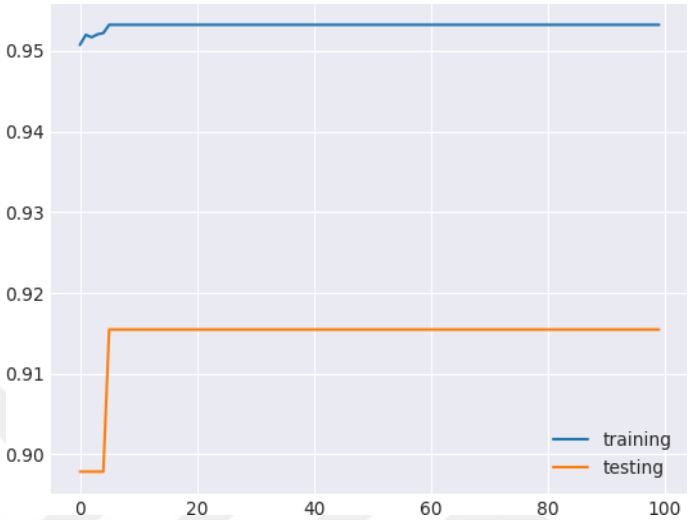

Figure 6.7 Minimum accuracy in each iteration

Figure 6.7 shows the minimum accuracy in each iteration. Genetic algorithm increases the accuracy value for both training and testing dataset.

The figures we mention above all show that genetic algorithm increases the values in each iteration and it is obvious that genetic algorithm converges very quickly in fifth iteration. Another way to investigate iterations of genetic algorithm can be investigating the standard deviation of the values, since we should expect the standard deviation to decrease in while the genetic algorithm converges. Figures 6.8 and 6.9 show the standard deviation of the metrics for training and testing dataset, respectively.

We can see similar patterns in Figures 6.8 and 6.9 where it shows that standard deviation of metrics decreases and it becomes zero at fifth iteration. If standard deviation is zero, we can understand that all values are the same and another

conclusion is that after fifth iteration, genetic algorithm stops changing the selected base models or the changing the base models does not effect the results anymore.

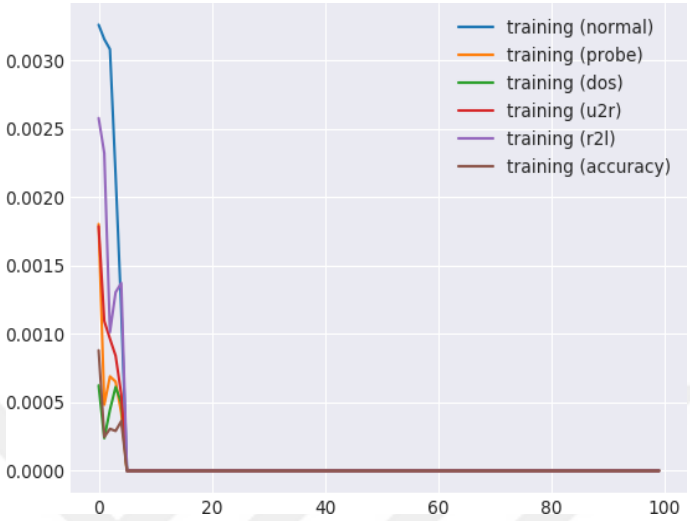

Figure 6.8 Standard deviation of the metrics in each iteration for training dataset

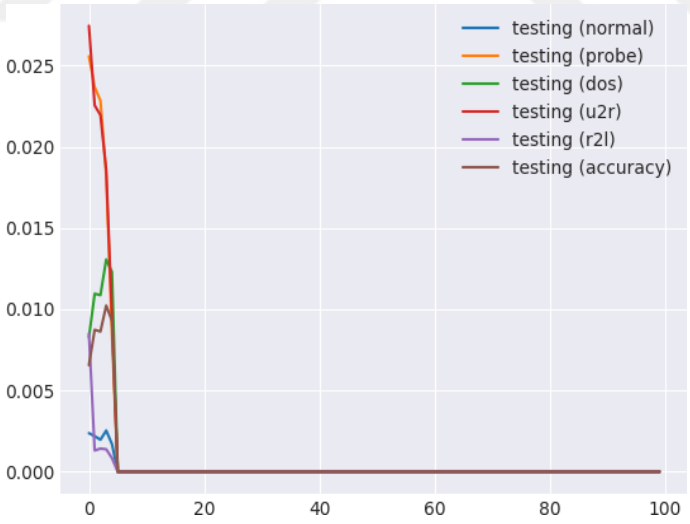

Figure 6.9 Standard deviation of the metrics in each iteration for testing dataset

After 100 iterations, genetic algorithm provided a combination of base models, which can be used as the final result. Even though, the number of iterations was 100, we need to mention that it converged at the fifth iteration.

Table 6.2 The results of genetic algorithm based experiment

normal	probe	dos	u2r	r2l	accuracy
96.75%	77.48%	96.23%	71.42%	9.98%	91.54%

Table 6.2 shows the results of the experiment discussed in this section. In section 5, we used the multiple metrics and used the best results of the experiments in Table 5.1 and 5.2. So, it is hard to come up a single solution for the study we discussed in section 5. In this current study, we have only one single result and the results of this study is shown in Table 6.2.

In Table 6.2 , we can see that u2r and r2l true positive rate has the highest score in the studies that we discussed in the scope of this thesis. This is a direct result of using oversampling method. While genetic algorithm helped us to come up with a single solution, over sampling helped us to come up with higher true positive rates for minor labels in exchange of about 1% decrease in overall accuracy.

Since we used naïve bayes as a combiner method in this study, we also compared the results of the previous study's results with the current result. Table 6.3 shows the comparison of the genetic algorithm based experiments and the model selection methods (with combiner naïve bayes) used discussed in section 5. Table 6.4 shows the comparison of the true positive rates of the same experiments.

Table 6.3 The comparison of experiments (accuracy)

Combiner	Model selection	Accuracy (%)
Naïve Bayes	all	89.19
	accuracy score	92.29
	information gain	92.39
	recall mean	91.90
	genetic algorithm	**91.54**

As we discussed above and as it is shown in Table 6.3 and 6.4, using genetic algorithm with oversampling increased the true positive rates of minor labels in exchange of decreasing the accuracy in 1%.

Table 6.4 The comparison of experiments (true positive rates)

Model selection	normal	probe	dos	u2r	r2l
all	97.3	78.9	93.2	47.1	6.0
accuracy score	97.9	80.2	97.2	30.0	5.7
information gain	97.9	79.9	97.2	30.0	7.6
recall mean	98.0	79.0	96.7	35.7	5.6
genetic algorithm	**96.75**	**77.48**	**96.23**	**71.42**	**9.98**

CHAPTER SEVEN
CONCLUSION

7.1 Conclusion

In this thesis, we have designed and implemented the stacking ensemble method to detect network intrusion and applied it on KDD'99 dataset. KDD'99 dataset provides an unbalanced dataset which makes the problem harder and which forces the researches to apply different methods.

In our first study which we explained in section 5, we did our experiments with different model selection methods and we used different metrics to select the base models. We saw that this method increases the performance when it is compared to a conventional machine learning algorithm (logistic regression) and when this approach is compared to other studies, we have seen better true positive rates for minor labels. Even though, this approach provided promising results, the problem with this method was using the best of the experiments in the comparison table. We also had to define some parameters which also may affect the performance of this method. Those problems forced us to improve this method with genetic algorithm.

In our second study which we explained in section 6, we used genetic algorithm as a base model selection process for stacking method. This helped the study to output only one single model. We used multi-objective genetic algorithm and so we could provide true positive rates of each labels, so genetic algorithm tried to optimize the all true positive rates at the same time.

The other improvement in the second study was using an over sampling method, which significantly improved the true positive rates for minor labels. The true positive rates for u2r and r2l were 47.1% and 7.6% in the first study and those results increased to 71.42% and 9.98%, respectively.

7.2 Future Work

As we described the in previous sections, when we provide results we give exact numbers; for example we say “we increased u2r from 47.1% to 71.42%”. Using bayesian statistics (not to be confused with naïve bayes classification) would calculate the uncertainties and calculate the credible intervals. Another possible future work would be using bayesian model selection as the selection process of base models for stacking ensemble method.

REFERENCES

Acharya, H. B., Kumar, S., Wadhwa, M., & Shah, A. (2016). Rules in play: On the complexity of routing tables and firewalls. In *Network Protocols (ICNP), 2016 IEEE 24th International Conference on,* 1-10.

Agarwal, R., & Joshi, M. V. (2001). PNrule: A new framework for learning classifier models in data mining (a case-study in network intrusion detection). In *Proceedings of the 2001 SIAM International Conference on Data Mining,* 1–17.

Al-Yaseen, W. L., Othman, Z. A., & Nazri, M. Z. A. (2015). Intrusion detection system based on modified k-means and multi-level support vector machines. In *International Conference on Soft Computing in Data Science,* 265–274.

Amor, N. B., Benferhat, S., & Elouedi, Z. (2004). Naive bayes vs decision trees in intrusion detection systems. In *Proceedings of the 2004 ACM symposium on Applied computing,* 420–424.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory,* 144–152.

Bosin, A., Dessì, N., & Pes, B. (2005). Intelligent Bayesian classifiers in network intrusion detection. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems,* 445–447.

Bouzida, Y., & Cuppens, F. (2006a). Detecting known and novel network intrusions. In *IFIP International Information Security Conference,* 258–270.

Bouzida, Y., & Cuppens, F. (2006b). Neural networks vs. decision trees for intrusion detection. In *IEEE/IST Workshop on Monitoring, Attack Detection and Mitigation (MonAM), 28*, 29-36

Breiman, L. (1996). Stacked regressions. *Machine Learning*, *24,* 49–64.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*, 5–32.

Bryll, R., Gutierrez-Osuna, R., & Quek, F. (2003). Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets. *Pattern Recognition*, *36*, 1291–1302.

Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, *2*, 121–167.

Cannady, J. (1998). Artificial neural networks for misuse detection. In *National information systems security conference*

Chawla, N. V. (2003). C4. 5 and imbalanced data sets: investigating the effect of sampling method, probabilistic estimate, and decision tree structure. In *Proceedings of the ICML, 3,* 66

Chebrolu, S., Abraham, A., & Thomas, J. P. (2005). Feature deduction and ensemble design of intrusion detection systems. *Computers & Security*, *24*, 295–307.

Cherkauer, K. J. (1996). Human expert-level performance on a scientific image analysis task by a system using combined artificial neural networks. In *Working notes of the AAAI workshop on integrating multiple learned models*.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*, 273–297.

de Lacerda, E. G., De Carvalho, A., & Ludermir, T. B. (2000). Evolutionary optimization of RBF networks. In *Neural Networks, 2000. Proceedings. Sixth Brazilian Symposium,* 219–224.

Davis J. (2004), Architectural Overview of the TCP/IP Protocol Suite, https://technet.microsoft.com/en-us/library/bb726993.aspx

Debar, H., & Dorizzi, B. (1992). An application of a recurrent network to an intrusion detection system. In *Neural Networks, 1992. IJCNN., International Joint Conference, 2*, 478–483.

Demir, N., & Dalkılıç, G. (2018). Modifed stacking ensemble approach to detect network intrusion. *Turkish Journal of Electrical Engineering & Computer Sciences*, *26*, 418–433.

Denning, D. E. (1987). An intrusion-detection model. *IEEE Transactions on Software Engineering*, *2*, 222–232.

Depren, O., Topallar, M., Anarim, E., & Ciliz, M. K. (2005). An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks. *Expert Systems with Applications*, *29*, 713–722.

Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, *10*, 1895–1923.

Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, *40*, 139–157.

Dietterich, T. G., & Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, *2*, 263–286.

Duin, R. P., & Tax, D. M. (2000). Experiments with classifier combining rules. In *International Workshop on Multiple Classifier Systems,* 16–29.

Efron, B., & Tibshirani, R. J. (1994). *An introduction to the bootstrap*. CRC press.

Eskin, E., Arnold, A., Prerau, M., Portnoy, L., & Stolfo, S. (2002). A geometric framework for unsupervised anomaly detection. In *Applications of data mining in computer security,* 77–101.

Fern, X. Z., & Brodley, C. E. (2003). Random projection for high dimensional data clustering: A cluster ensemble approach. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03),* 186–193.

Fleizach, C., & Fukushima, S. (1998). A naive Bayes classifier on 1998 KDD Cup, http://sysnet.ucsd.edu/~cfleizac/cse250b/project1.pdf

Gharibian, F., & Ghorbani, A. A. (2007). Comparative study of supervised machine learning techniques for intrusion detection. In *Communication Networks and Services Research, 2007. CNSR'07. Fifth Annual Conference,* 350–358.

Guan, Y., Ghorbani, A. A., & Belacel, N. (2003). Y-means: A clustering method for intrusion detection. In *Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference, 2*, 1083–1086.

Gunes Kayacik, H., Nur Zincir-Heywood, A., & Heywood, M. I. (2007). A hierarchical SOM-based intrusion detection system. *Engineering Applications of Artificial Intelligence*, *20*, 439–451.

Gupta, G. P., & Kulariya, M. (2016). A framework for fast and efficient cyber security network intrusion detection using apache spark. *Procedia Computer Science*, *93*, 824–831.

Gyanchandani, M., Yadav, R., & Rana, J. (2010). Intrusion Detection using C4. 5: Performance Enhancement by Classifier Combination. In *Proceedings of the International Conference on Advances in Computer Science,* 130–133.

Hasan, M. A. M., Nasser, M., Pal, B., & Ahmad, S. (2014). Support vector machine and random forest modeling for intrusion detection system (IDS). *Journal of Intelligent Learning Systems and Applications*, *6*, 45.

Haykin, S., & Network, N. (2004). A comprehensive foundation. *Neural Networks.* NY: Prentice Hall.

Ho, T. K. (1998). The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*, 832–844.

Ho, T. K., Hull, J. J., & Srihari, S. N. (1994). Decision combination in multiple classifier systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *16*, 66–75.

Höglund, A. J., & Hätönen, K. (1998). Computer network user behaviour visualisation using self organising maps. In *ICANN 98,* 899–904.

Khan, L., Awad, M., & Thuraisingham, B. (2007). A new intrusion detection system using support vector machines and hierarchical clustering. *The VLDB Journal—The International Journal on Very Large Data Bases*, *16*, 507–521.

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

Kohonen, T. (2012). *Self-organization and associative memory, 8,* Springer Science & Business Media.

Kruegel, C., Mutz, D., Robertson, W., & Valeur, F. (2003). Bayesian event classification for intrusion detection. In *Computer Security Applications Conference, 2003. Proceedings. 19th Annual,* 14–23.

Leung, K., & Leckie, C. (2005). Unsupervised anomaly detection in network intrusion detection using clusters. In *Proceedings of the Twenty-eighth Australasian conference on Computer Science,* 38, 333–342.

li, W., & Liu, Z. (2011). A method of SVM with Normalization in Intrusion Detection. *Procedia Environmental Sciences*, *11*, 256–262.

Lippmann, R., Haines, J. W., Fried, D. J., Korba, J., & Das, K. (2000). The 1999 DARPA off-line intrusion detection evaluation. *Computer Networks*, *34*, 579–595.

Mitchell, T. (1997). Machine learning. *Burr Ridge, IL: McGraw Hill*, *45*, 870–877.

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,* 1702–1707.

Mutz, D., Valeur, F., Vigna, G., & Kruegel, C. (2006). Anomalous system call detection. *ACM Transactions on Information and System Security (TISSEC)*, *9*, 61–93.

Öztürk, N. (2003). Use of genetic algorithm to design optimal neural network structure. *Engineering Computations*, *20*, 979–997.

Pan, Z.-S., Chen, S.-C., Hu, G.-B., & Zhang, D.-Q. (2003). Hybrid neural network and C4. 5 for misuse detection. In *Machine Learning and Cybernetics, 2003 International Conference, 4, ,* 2463–2467.

Panda, M., & Patra, M. R. (2007). Network intrusion detection using naive bayes. *International Journal of Computer Science and Network Security*, *7*, 258–263.

Peddabachigari, S., Abraham, A., Grosan, C., & Thomas, J. (2007). Modeling intrusion detection system using hybrid intelligent systems. *Journal of Network and Computer Applications*, *30*, 114–132.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 2825–2830.

Peng, Y., Kou, G., Wang, G., & Shi, Y. (2011). FAMCDM: A fusion approach of MCDM methods to rank multiclass classification algorithms. *Omega*, *39*, 677–689.

Pfahringer, B. (2000). Winning the KDD99 classification cup: bagged boosting. *ACM SIGKDD Explorations Newsletter*, *1*, 65–66.

Portnoy, L., Eskin, E., & Stolfo, S. (2001). Intrusion detection with unlabeled data using clustering. In *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security.*

Özgür, A., & Erdem, H. (2016). A review of KDD99 dataset usage in intrusion detection and machine learning between 2010 and 2015. *PeerJ PrePrints*

Sabhnani, M., & Serpen, G. (2003). Application of Machine Learning Algorithms to KDD Intrusion Detection Dataset within Misuse Detection Context. In *MLMTA* (pp. 209–215).

Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., & Williamson, R. C. (2001). Estimating the support of a high-dimensional distribution. *Neural Computation*, *13*, 1443–1471.

Scott, S. L. (2004). A Bayesian paradigm for designing intrusion detection systems. *Computational Statistics & Data Analysis*, *45*, 69–83.

Sinclair, C., Pierce, L., & Matzner, S. (1999). An application of machine learning to network intrusion detection. In *Computer Security Applications Conference, 1999.(ACSAC'99) Proceedings. 15th Annual,* 371–377.

Skurichina, M., & Duin, R. P. (2005). Combining feature subsets in feature selection. In *International Workshop on Multiple Classifier Systems,* 165–175.

Smyth, P., & Wolpert, D. (1998). Stacked density estimation. In *Advances in neural information processing systems,* 668–674.

Song, J., Ohira, K., Takakura, H., Okabe, Y., & Kwon, Y. (2008). A clustering method for improving performance of anomaly-based intrusion detection system. *IEICE TRANSACTIONS on Information and Systems*, *91*, 1282–1291.

Thames, J. L., Abler, R., & Saad, A. (2006). Hybrid intelligent systems for network security. In *Proceedings of the 44th annual Southeast regional conference,* 286–289.

Ting, K. M., & Witten, I. H. (1999). Issues in stacked generalization. *J. Artif. Intell. Res.(JAIR)*, *10*, 271–289.

Toosi, A. N., & Kahani, M. (2007). A new approach to intrusion detection based on an evolutionary soft computing model using neuro-fuzzy classifiers. *Computer Communications*, *30*, 2201–2212.

Wang, Y., Wong, J., & Miner, A. (2004). Anomaly intrusion detection using one class SVM. In *Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC,* 358–364.

Witten, I. H., Frank, E., Hall, M. A., & Pal, C. J. (2016). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, *5*, 241–259.

Xu, L., Krzyzak, A., & Suen, C. Y. (1992). Methods of combining multiple classifiers and their applications to handwriting recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, *22*, 418–435.

Yao, X. (1993). A review of evolutionary artificial neural networks. *International Journal of Intelligent Systems*, *8*, 539–567.

Zhang, J., & Zulkernine, M. (2006). A hybrid network intrusion detection technique using random forests. In *Availability, Reliability and Security, 2006. ARES 2006. The First International Conference,* 8 - 18

Zhou, Z.-H. (2012). *Ensemble methods: foundations and algorithms*. NY: CRC press.

Appendix 1: KDD'99 Dataset

Table A-1.1 Features of KDD'99 Dataset

Feature Name	Type
duration	Numeric
protocol_type	categorical
service	categorical
flag	categorical
src_bytes	numeric
dst_bytes	numeric
land	categorical
wrong_fragment	numeric
urgent	numeric
hot	numeric
num_failed_logins	numeric
logged_in	categorical
num_compromised	numeric
root_shell	numeric
su_attempted	numeric
num_root	numeric
num_file_creations	numeric
num_shells	numeric
num_access_files	numeric
is_host_login	categorical
is_guest_login	categorical
count	numeric
srv_count	numeric
serror_rate	numeric
srv_serror_rate	numeric
rerror_rate	numeric
srv_rerror_rate	numeric
same_srv_rate	numeric
diff_srv_rate	numeric
srv_diff_host_rate	numeric
dst_host_count	numeric
dst_host_srv_count	numeric
dst_host_same_srv_rate	numeric
dst_host_diff_srv_rate	numeric
dst_host_same_src_port_rate	numeric
dst_host_srv_diff_host_rate	numeric
dst_host_serror_rate	numeric
dst_host_srv_serror_rate	numeric
dst_host_rerror_rate	numeric
dst_host_srv_rerror_rate	numeric

Table A-1.2 First 5 Rows of 10% of KDD Dataset

0,tcp,http,SF,181,5450,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,9,9,1.00,0.00,0.11,0.00,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,239,486,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,19,19,1.00,0.00,0.05,0.00,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,235,1337,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,29,29,1.00,0.00,0.03,0.00,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,219,1337,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,6,6,0.00,0.00,0.00,0.00,1.00,0.00,0.00,39,39,1.00,0.00,0.03,0.00,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,217,2032,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,6,6,0.00,0.00,0.00,0.00,1.00,0.00,0.00,49,49,1.00,0.00,0.02,0.00,0.00,0.00,0.00,0.00,normal.

Table A-1.3 First 5 Rows of Corrected KDD Dataset

0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,254,1.00,0.01,0.00,0.00,0.00,0.00,0.00,0.00,normal.
0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,254,1.00,0.01,0.00,0.00,0.00,0.00,0.00,0.00,normal.
0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,254,1.00,0.01,0.00,0.00,0.00,0.00,0.00,0.00,normal.
0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,254,1.00,0.01,0.00,0.00,0.00,0.00,0.00,0.00,snmpgetattack.
0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,254,1.00,0.01,0.01,0.00,0.00,0.00,0.00,0.00,snmpgetattack.

Appendix 2: Development Environment

- OS: Linux
- Editor: PyCharm
- Programming language: Python
- Machine learning library: Scikit
- The library for manipulating data: Pandas
- The library we used for converting data to matrix: patsy
- The library to implement genetic algorithm: deap
- The library to do numeric operations on data pre-processing phase: numpy
- The library to store the machine learning models: pickle
- Hardware: 16GB Memory + 4 CPUs
    - We used an Amazon EC2 instance: m5.xlarge (which has 4 CPUs and 16 GB Memory)
    - The study we described in Chapter 5 took approximately 24 hours to run
    - The study we described in Chapter 6 took approximately 18 hours to run
    - Both of the studies consumed all the 4 CPUs by 99% on the training phase except the process of loading data. We used only one CPU for prediction phase and the usage was 99% for one CPU on that prediction phase.
    - Both of the studies consumed the memory by 80% (in average) on the training phase and the prediction phase.

Appendix 3: Confusion Matrices of 13 Experiments

Table A-3.1 Confusion Matrix of Experiment #1.

	normal	probe	dos	u2r	r2l	true positive rate
normal	59450	170	952	3	18	98.11%
probe	1123	2864	179	0	0	68.75%
dos	39502	7	190344	0	0	82.81%
u2r	56	1	1	6	6	8.57%
r2l	15400	17	7	1	922	5.64%

Table A-3.2 Confusion Matrix of Experiment #2.

	normal	probe	dos	u2r	r2l	true positive rate
normal	59572	233	778	4	6	98.31%
probe	431	3477	257	0	1	83.46%
dos	6182	83	223584	0	4	97.27%
u2r	48	1	5	9	7	12.86%
r2l	15487	6	3	3	848	5.19%

Table A-3.3 Confusion Matrix of Experiment #3.

	normal	probe	dos	u2r	r2l	true positive rate
normal	59626	199	754	0	14	98.40%
probe	930	3038	197	0	1	72.92%
dos	6229	20	223604	0	0	97.28%
u2r	51	0	6	6	7	8.57%
r2l	15428	1	5	1	912	5.58%

Table A-3.4 Confusion Matrix of Experiment #4.

	normal	probe	dos	u2r	r2l	true positive rate
normal	59630	193	756	0	14	98.41%
probe	892	3070	204	0	0	73.69%
dos	6204	12	223637	0	0	97.30%
u2r	52	0	6	5	7	7.14%
r2l	15426	1	4	0	916	5.60%

Table A-3.5 Confusion Matrix of Experiment #5.

	normal	probe	dos	u2r	r2l	true positive rate
normal	59476	254	851	1	11	98.16%
probe	942	3014	210	0	0	72.35%
dos	6154	12	223687	0	0	97.32%
u2r	50	0	7	6	7	8.57%
r2l	15418	5	5	0	919	5.62%

Table A-3.6 Confusion Matrix of Experiment #6.

	normal	probe	dos	u2r	r2l	true positive rate
normal	60255	224	97	1	16	99.44%
probe	779	3208	167	0	12	77.00%
dos	5783	227	223721	0	122	97.33%
u2r	48	1	2	10	9	14.29%
r2l	15909	17	11	8	402	2.46%

Table A-3.7 Confusion Matrix of Experiment #7.

	normal	probe	dos	u2r	r2l	true positive rate
normal	59981	205	388	0	19	98.99%
probe	947	3071	95	0	53	73.72%
dos	6212	38	223603	0	0	97.28%
u2r	47	4	7	5	7	7.14%
r2l	15357	3	69	0	918	5.62%

Table A-3.8 Confusion Matrix of Experiment #8.

	normal	probe	dos	u2r	r2l	true positive rate
normal	59988	208	381	0	16	99.00%
probe	790	3260	112	0	4	78.25%
dos	6159	28	223665	0	1	97.31%
u2r	49	0	6	6	9	8.57%
r2l	15395	21	5	0	926	5.66%

Table A-3.9 Confusion Matrix of Experiment #9.

	normal	probe	dos	u2r	r2l	true positive rate
normal	59987	192	402	5	7	99.00%
probe	800	3133	231	0	2	75.20%
dos	6116	175	223556	0	6	97.26%
u2r	47	0	2	13	8	18.57%
r2l	15394	22	2	36	893	5.46%

Table A-3.10 Confusion Matrix of Experiment #10.

	normal	probe	dos	u2r	r2l	true positive rate
normal	58940	201	692	719	41	97.27%
probe	188	3287	107	584	0	78.90%
dos	6925	41	214157	8730	0	93.17%
u2r	29	0	0	33	8	47.14%
r2l	14920	1	1	443	982	6.01%

Table A-3.11 Confusion Matrix of Experiment #11.

	normal	probe	dos	u2r	r2l	true positive rate
normal	59291	228	58	982	34	97.85%
probe	431	3342	154	235	4	80.22%
dos	6095	33	223463	258	4	97.22%
u2r	38	0	4	21	7	30.00%
r2l	15141	0	1	268	937	5.73%

Table A-3.12 Confusion Matrix of Experiment #12.

	normal	probe	dos	u2r	r2l	true positive rate
normal	59305	208	59	973	48	97.87%
probe	449	3330	155	232	0	79.93%
dos	6099	25	223474	255	0	97.22%
u2r	36	0	4	21	9	30.00%
r2l	14840	0	1	265	1241	7.59%

Table A-3.13 Confusion Matrix of Experiment #13.

	normal	**probe**	**dos**	**u2r**	**r2l**	**true positive rate**
normal	59406	241	713	200	33	98.04%
probe	571	3293	67	234	1	79.04%
dos	7265	55	222190	343	0	96.67%
u2r	38	0	0	25	7	35.71%
r2l	15026	15	1	382	923	5.65%