UiO : Department of Informatics
University of Oslo

Anomaly detection using machine learning techniques

A comparison of classification algorithms

Henrik Hivand Volden

Master's Thesis Spring 2016

Anomaly detection using machine learning techniques

Henrik Hivand Volden

May 23, 2016

Abstract

Machine learning is an emerging science that is implemented in many different technologies and in many ways. Researchers have been looking into how machine learning could be used in network security to do the same or a better job than solutions available today. For this idea to be further developed there has to be some proof-of-concepts or studies which indicates that this idea is possible to develop.

This thesis aims to implement anomaly detection using machine learning techniques. The algorithms used are k-NN and SVM and the implementation is done by using a data set to train and test the two algorithms. The data set used in this thesis is the improved version of the KDD CUP99 data set, named NSL-KDD. There are measured several parameters and metrics in order to determine which of the two implemented algorithms is more suited for anomaly detection in a network environment.

The results gained in this thesis indicated that the algorithm k-NN is more suited for anomaly detection using machine learning techniques, than SVM. Further investigations has to be done in order to confirm and improve the results from the solutions implemented this thesis.

Acknowledgements

I would like to express my sincere gratitude and appreciation to:

- My supervisor Desta Haileselassie Hagos, for his support and help throughout the project.
- The rest of the professors that have advised and helped me during the whole master program.
- My fellow master students for all the help, motivation and encouragement during the two years.
- Last but not least, I would like to express my deepest gratitude and appreciation to my family for their unconditional love and support.

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Introduction

Security is a growing industry in today's labor market, it has become a very important part of businesses to have good security teams and solutions. Not only are the security teams getting bigger and better, but the level of security that businesses apply are getting more complex. The reason for this is also the fact that the threats are getting more evolved and complicated [1]. *Symantec* and many other companies confirm this in their security reports of the year and they are expecting to grow for each year in the nearest future [1, 2, 3].

The problem with the evolution of threats and attacks is that they are getting harder to detect and therefore it could be difficult to find out if network traffic is legit or malicious. Intrusion detection systems (IDS's) are doing a decent job in detecting malicious traffic but a standard IDS must continuously be updated with rule-sets and upgrades to be up-to-date with the recent threat vectors. The biggest companies like e.g. Norton releases new rule-sets on a regular basis, but even these might not be sufficient. How is it possible to keep up and be up-to-date on this, when even the rules published might not be sufficient. This is a subject that many security researchers has inquired.

Some researchers have been trying to study the field of machine learning, to get a grasp on this issue. Using machine learning could be an interesting step in the evolution of security software. The reason for using machine learning is that it could help to automate the handling of threats, and keep the system up-to-date by analyzing threats and recognize them. When using machine learning, the software that is used is trained to recognize traffic pattern so that it will be able to classify the different events and deny or allow the traffic. It classifies the traffic in the way that it is trained to recognize the patterns by using a big training data set, containing different attacks.

In this paper the focus will be on using machine learning techniques to do anomaly detection. This is going to be done by using a training- and test-data set, that is labeled, and contains several attack types. These different attacks are what is going to be attempted to detect and recognized using different algorithms.

1.1 Problem statement

Comparing the two algorithms SVM and k-NN, to determine which of the algorithms that yields the better overall performance, when doing anomaly detection using machine learning techniques

The parameters that determines which of the algorithms that has the best overall performance is:

- Classification performance
- Time consumption
- Resource consumption

Chapter 2

Background

This chapter will contain information about the different technologies, terms and relevant studies that has already been done in the field. The focus in this chapter is to give a brief introduction to software and technologies needed to understand the problem that is going to be explored later in the project.

2.1 Intrusion Detection System(IDS)

Intrusion Detection Systems are used to recognize suspicious traffic in a computer network. An IDS can work in different ways, but are often built on the same ground pillars, which could be misuse or anomaly detection.

In most cases an IDS will look for signatures or irregularities in the systems traffic based on written rule-sets or measurements done. These measurements are supposed to be a normalized state for the network or system, and if there are any traffic that is not recognized as normal it is alarmed. These irregularities are often called malicious/suspicious traffic, but it could also be traffic that is not malicious. What most IDSs such as Snort[1], Suricata[2] etc. does is to filter the traffic based on one or more rule-sets [4].

If the IDS is setup to block or drop the traffic it is no longer called an IDS, then it becomes an Intrusion Prevention System(IPS). An IPS works just as an IDS, but instead of just alarming the suspicious traffic it can block or drop traffic that is marked as malicious.

2.1.1 Misuse(signature/rule) Based Detection

Misuse or signature/rule based detection is when there are written rule-sets applied to detect the malicious traffic. These rules can be written based on known facts like IP addresses, content in payloads, URLs etc. There is many standard rule-sets available, and most of the software comes

[1] https://www.snort.org/
[2] https://suricata-ids.org/

with some basic rules, but to be able to keep up with the latest threats, these need to be updated on a regular basis [5, 6].

Typical software that uses misuse signature based detection is *Snort and Suricata*. These two software are widely used all over the world, and are very effective when used correct and kept up-to-date. For businesses to be able to do all of this, there have to be used a lot of time and money. Once it is set up there has to be someone monitoring the different alerts, to check for false positives and true positives. False positives means that there has been alerted a potential threat to the system, which actually are not a threat after a further investigation. True positive means that there alerted possible threat, is actually a threat.

2.1.2 Anomaly Based Detection

Anomaly based detection is a way to detect irregular behavior in the network or in the traffic within a network. How the IDS knows that there is irregular traffic is based on measurements done prior to deployment of the IDS. This tests or training-sets are used to simulate the traffic that is expected to be "normal" within the network environment. These measurements are then the basis for what "normal" traffic should look like, and if the traffic deviates from the "normal traffic" there will be generated alerts based on the traffic. The training-sets are also used to simulate malicious traffic so that it will recognize the patterns from known threats and attacks [5, 6].

2.2 Machine Learning

2.2.1 Definitions

Machine Learning was defined already in 1959 by Arthur Samuel as: *"field of study that gives computers the ability to learn without being explicitly programmed"* [7, 8] Arthur Samuel is mentioned by many researchers that writes or talks about machine learning, and he is considered to be one of the first founders of it. He wanted to use machine learning to make a computer *"learn"* to play checkers.

When Samuel started out it was just on an algorithmic basis and it took some years of developing before he had implemented a program that actually worked. He implemented a *"Tree"* structure to do the decision making. The *"Tree"* structure calculates a few moves ahead based on statistics. These statistics are again based on probabilities of what the next move is likely to be. The actual setup of the software that does the ranking of what the next move might be, is trained many times so that it will know what moves that is possible and what would be the optimal move next. It also takes into consideration what moves that can be done by the opponent, so it will *"think"* what the next few moves that follows might be [7, 9].

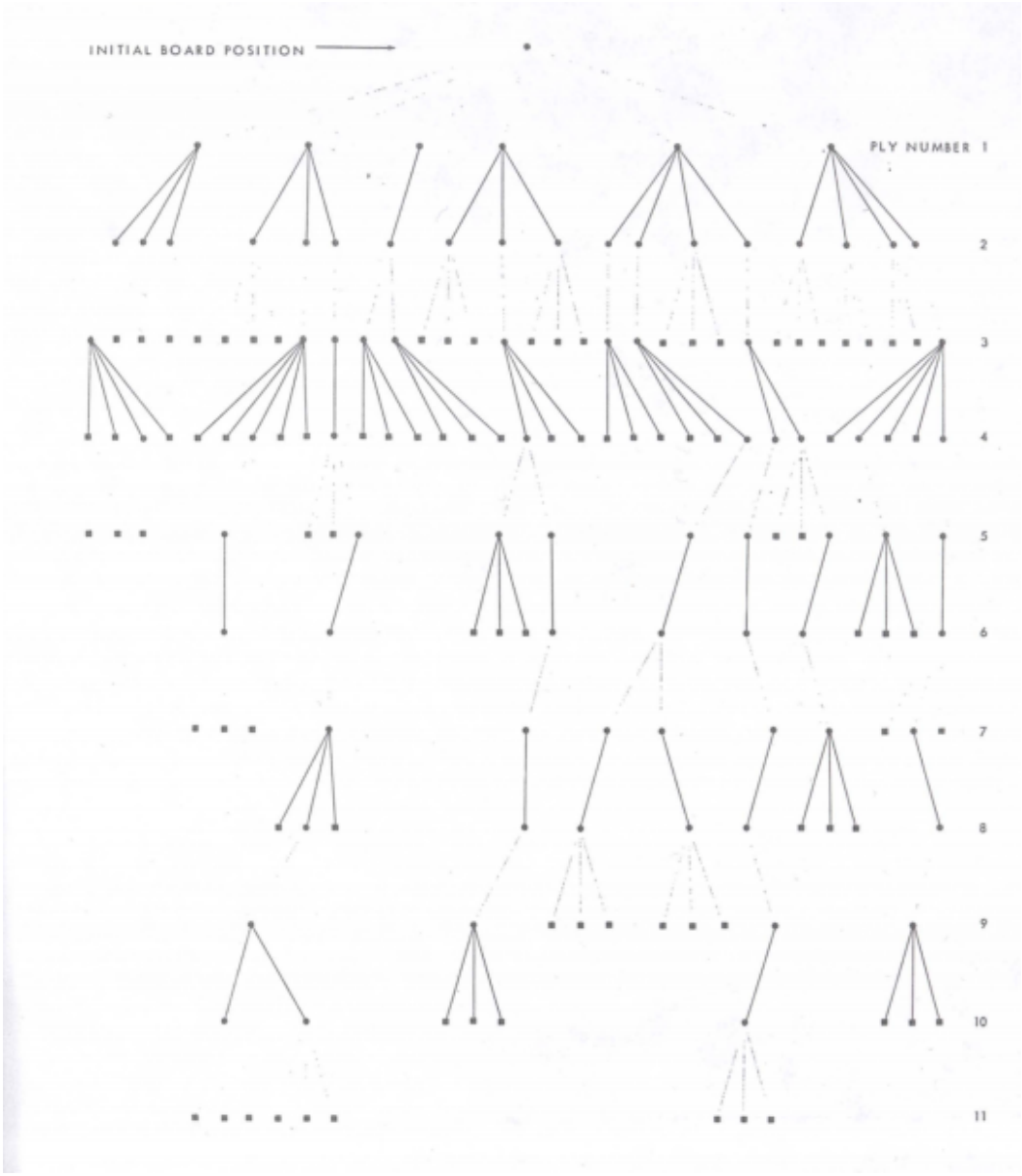


Figure 2.1: This figure displays the sketch used by Arthur Samuel in the paper that he wrote about Checkers and Machine learning [7].

Figure 2.1 displays the idea behind the *"Tree"* structure used as sort of an algorithm. Samuel developed his own algorithm due to the fact that he could not find any existing algorithm that would work in a matter as good as he wanted it to.

Later in 1998 Tom Michael Mitchells came up with a "new" definition. Mitchells definition is a more formal and mathematical explanation. It is quoted by many researchers since it is a little more scientific in the way it is presented. The definition he came up with was:*"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E"* [10, 11].

Machine learning is a research-field often considered to be under Artificial Intelligence(AI). This is due to the fact that the machine should be able to make decisions, taking into consideration statistics and known facts when deciding. This is a way of giving a machine human capabilities and it is considered to be AI [12]. AI is a wide term and grasps a lot of different technologies. Machine learning is just one of them, there is also technologies like genetic programming, artificial neural networks etc.

Machine learning is often also considered to be part of data mining, this is sort of correct since data mining can use machine learning tools and techniques but it lacks an important part of the machine learning features which is the decision making [13]. There are of course some exceptions where data mining also make decisions, but it is mostly used to calculate probabilities and statistics and that is it. There are examples like automatic stock trading, insulin pumps, pacemakers etc. these are examples that could be called data mining and machine learning since there is a decision and some calculations done.

2.2.2 How Machine learning is used today

Machine learning is used to do many different things in today's society. It is used to do important calculations like whether or not to inject more insulin into a person suffering from diabetes through a insulin pump etc. There is so many things that it can be used to so under is a list containing some of the areas that is used [14].

- Fraud detection.
- Web search filtering.
- Real-time ads.
- Text analysis.
- Facebook uses it for its news feed and the chat column.
- Pattern and image recognition.
- Email spam filtering.
- Health care appliances
- Network intrusion detection.

As can be seen from the list above there is a wide area that machine learning is used and it keeps on and gets bigger as machines and technologies are getting better and faster.

2.2.3 Machine Learning techniques

Machine learning techniques is different approaches to how machine learning could be used and how this is done. There are several ways of doing machine learning and below the different techniques will be explained in more detail [11].

2.2.4 Supervised Learning

Supervised learning is an machine learning technique, supervised is as the word says that there is some sort of assistants in the way that it is used. There could be that the way of detecting is assisted by labels that marks the data that is going to be used. This could be used e.g. if someone is trying to detect the most common or least common color for cars using a traffic camera. Then they could label the different colors of the cars and match those colors too the cars on the traffic camera. This is just one example where supervised learning is used. Supervised learning makes the process of detection or decision making easier, since there are pointers saying that for example the car is a specific color or that the traffic is a type of attack. If the data that it going to be used is labeled, the machine learning technique used is supervised learning [14, 11].

2.2.5 Unsupervised Learning

Unsupervised learning is the opposite of supervised learning, instead of just inspecting one of the labels, which is done in supervised, it looks at the whole picture. It gathers information and processes the whole input, in order0 to make decisions based on the entire input. When using unsupervised learning the system does not know the right "answers". This is one of the main purposes off using unsupervised learning, it should be able to take in the input that is sent in/given and come up with some structured answer based on patterns that it had been trained to recognize or to gather groups of information. It could e.g. be used by stores to analyze patterns in peoples shopping habits and group the people with the same habits and send out offers based on that [14, 11].

2.2.6 Algorithms

What is algorithms and how can they be used with machine learning? When doing machine learning there is a need for an classification algorithm. This algorithm is then used to classify and decide what to do with the information gathered or at hand. So the algorithm is what the machine learning sort of uses a way of "thinking". There are many different algorithms with several different purposes, and algorithms can be used for many different purposes [15].

In this paper algorithms is going to be used as the decision maker for anomaly detection using machine learning techniques. The way that is going to be done is by applying an algorithm to the data set and compare the results of the two algorithms to see which of the two who have the best results.

Support Vector Machine (SVM)

SVM or Support Vector Machine is an algorithm that is very often used when applying machine learning techniques to do anomaly detection. It is an algorithm that is easily applied and could give good results if used in a correct manner. It is a classification algorithm that uses hyperplane classifiers [16, 17, 18]. The hyperplane is calculated so that it would separate the different classes. This could be confusing to understand, just to clarify it, say that there are two classes. The two classes are boxes and circles as can be seen on figure 2.2.

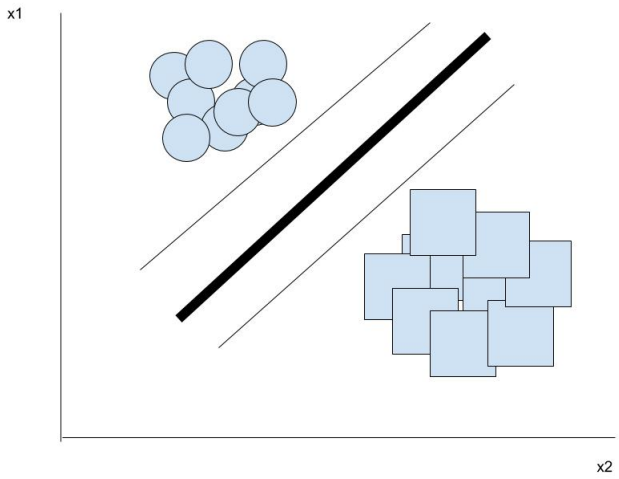


Figure 2.2: This figure displays how the SVM algorithm works with two binary separable binary sets, which here is represented using boxes/squares and circles.

When looking at figure 2.2 the hyperplane is the 3 line in between the circles and the boxes. The one in the middle is the hyperplane and the two others is what often is referred to as the *error margin* or just *margin*. If there are any data points within this error margin, the points within could be classified as the opposite of what it should have been. As illustrated in figure 2.2 there is no data points within the margin and therefore all of the circles will be classified as x1 and all of the boxes will be classified as x2. This is how the SVM algorithm works, explained in a very simple manner to make it more easy to grasp [16, 17, 18].

k-Nearest Neighbour (k-NN)

K-Nearest Neighbour is known as one of the simplest and most fundamental classifications algorithms. It is a widely used algorithm due to the fact that it so easy to use and learns complex functions easily. In many of the papers published about supervised learning, k-NN is a very common algorithm because of it simplicity and that it could be applied to different types of data [19, 20, 21].

The way k-NN classify an object is that it looks at the k-Nearest Neighbours and the majority vote of the nearest neighbours determines what class to classify the object to. If i.e., k=2 it will look at the two nearest neighbours and determine what to classify the object to by looking at the two nearest neighbours.

To make the algorithm more understandable a graphical illustration is made, to make it a little more clear on how it actually works when applied. Say e.g. that there are two different types of data that is going to be classified. In figure 2.3 this is illustrated using X-symbols and circles. What is going to be classified is the green diamond in the middle. If k=2 there will be a tie between the two classes it could be classified as. Since there is one of each class that is the two nearest neighbours. It will then be classified as the one which is the closest the diamond. If k=3 there is a majority vote where it is 2 circles against one X-symbol, if this was the case the diamond would be classified as X2 which is circles.

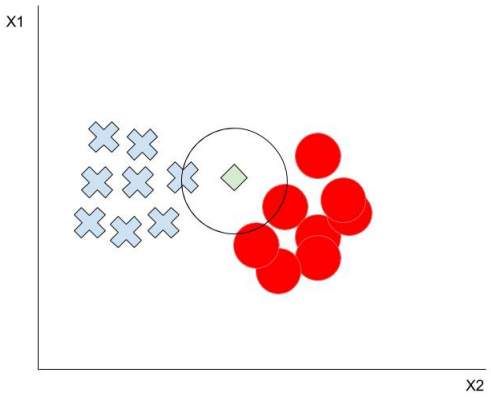


Figure 2.3: This figure displays how the k-NN algorithm works with two binary separable binary sets, which here is represented using X-symbols and circles. The diamond in the middle is what is going to be classified.

The k-NN classifier is commonly based on the Euclidean distance, which is the between a test sample and a specified training sample. This could also be explained by using an equation, it would then be like this [19]:

$$d(x_{i,1}, x_{l,1}) = \sqrt{(x_{i,1} - x_{l,1})^2 + (x_{i,2} - x_{l,2})^2 + \ldots + (x_{i,p} - x_{l,p})^2} \quad (2.1)$$

The equation 2.1 explains how the Euclidean distance is calculated, which is what the k-NN is commonly based on. There is also some other metrics such as Overlap metric or Hamming distance. Using k-NN on a data set with many variables that is much alike will make the algorithm have a worse accuracy than if there is much variance in the data sets variables. So one could say that it is an algorithm that works good with many different inputs (data sets, data-streams etc.) [19, 20, 21].

2.3 KDD cup99

KDD cup99 data set was used at *The Third International Knowledge Discovery and Data Mining Tools Competition*. The purpose of using the data set then, was to create an intrusion detector, a predictive model that could determine whether the traffic was *"bad"* or *"good"* [22]. Since this competition the data set has been used by many researchers to practice, train, test and implement machine learning techniques.

This data set has been used by many researchers since it was publicly available, and contained a lot of data points. One of the many reasons that this is done is due to the fact that all of the traffic is labeled as malicious or normal traffic. The data points that are malicious is tagged with the kind of an attack it is supposed to simulate at the end of line in each line of traffic. Each line in the data set is an entry and contains the same data one would find when looking at logs off network traffic, but as said it contains an extra column that says whether or not it is malicious [22].

The KDD cup99 data set has for a long time been considered to be a very good data set when training algorithms [23]. One of the reasons for that is due to it large amount entries and that it is labeled traffic. This is one of the premises for using supervised learning that the data is labeled and the machine actually knows what kind of threats it should recognize. In the data set we find multiple attacks and groupings of the attacks [22]:

- DoS(Denial of Service)
  - Back
  - Land
  - Neptune
  - Pod
  - Smurf
  - Teardrop
- U2R(User too root)
  - Buffer overflow
  - Loadmodule
  - Perl
  - Rootkit
- R2L(Root too local)
  - Ftp write
  - Guess passwd
  - Multihop
  - Phf

  - Imap
  - Spy
  - Warezclient
  - Warezmaster
- Probe
  - Ipsweep
  - Nmap
  - Portsweep
  - Satan

These are all of the attacks in the data set grouped by types of attacks.

The KDD CUP99 has for a long time been considered to be a good data set to train algorithms and to test with, but in the later years many researchers like S Terry Brugger et. al.[24] has criticised the data set for giving false results due to errors, redundant etc. data in the data set.

Another important fact about the data set is that it was created for a competition and it was then "handcrafted" for the purpose. What is meant by this is that the machine that were set up to monitor traffic and log the different types of traffic, was expecting some "handcrafted" packets/traffic. Many researchers has later criticised how this was done, but it has been kept as one of the most used data set when doing anomaly detection using machine learning techniques [24, 25, 26].

The environment that were "attacked" was intended to simulate the same amount of traffic as what an medium sized military base in the US could be exposed to over a period of time. There is also some issues with the TTL and how this is used in the data set. Normaly a TTL would be different depending on several different factors, but in the original data set this either 126 or 253 for the malicious traffic and 127 or 254 for the normal traffic [24, 25, 26, 27].

2.3.1 NSL-KDD

The NSL-KDD data set is an improvement of the old KDD CUP99 data set and is what is going to be used in this thesis. It has been improved by removing some of the redundant data points, that could cause errors and give better results than what should be [28].

There has also been stated that the KDD original data set had some inherent problems, these are removed as well as the other issues so the data set though it is smaller than the original data set, the NSL-KDD should be an improvement and should be more exact in the way it is created and most of the issues mentioned in the criticism of the original data set should be addressed.

2.4 Rstudio

Rstudio is the client web GUI, that runs Rserver as a back-end is a programming platform for the programming language R. R is very often used as a statistical calculation tool, this is due to all of the possibilities built in to R with all of the packages available for it. It could be used in a good way to make raw data more understandable by plotting, calculating key numbers such as means, median, min-, max-values, etc. [29].

What R also could be used to are: read and understand big data set like KDD CUP99 and make sense out of the data by i.e. applying algorithms to work as an detection mechanism, reading the data into a data frame or an array so one could be able to use it to calculate or register different key values or features. There are many possibilities using R. But in this thesis R is going to be used as platform to develop and apply algorithms to calculate or compute the outcome of applying two different algorithms using machine learning techniques [29].

2.5 Related works

2.5.1 Paper 1:

The first paper that can be related to this thesis work is a paper written by Mulay et. la.[30]. It is written in 2010 and looks into IDS's using SVM and decision tree. In this paper they used machine learning techniques to apply SVM and decision tree algorithm to the KDD CUP99 data set and got god results. They also used SVM and decision tree combined to see which would get the best results. The way they used decision tree was that they used a decision tree based SVM. It is stated in the paper that this is a good way of solving it, when working with multiclass data [30].

2.5.2 Paper 2:

In the second paper found in the field of this thesis is written by X. Xu et. la. [31] the motivation is to optimize the training of the classifier and the tests. The optimization they experimented with was an algorithm called PCA(Principal Component Analysis). What the PCA does is that it gets rid of the unnecessary data that comes in to the IDS and then it runs SVM on the data [31].

The data that were used to simulate traffic was the KDD CUP 99 data set. And the result could be interpreted as both positive and negative due to it runs much faster using the PCA but the results with the PCA running gives a worse classification performance or detection rate if you want. In the paper they state that there were not much of a difference with or without the PCA when it comes to classification performance, but when looking at the numbers the best results is without running the PCA algorithm [31].

2.5.3 Paper 3:

The third paper that has been looked into is a paper about using clustering to improve the KNN-based classifiers for online anomaly network traffic identification[32]. It is written by Ming-Yang Su et. la.[32]. In this paper the researchers has chosen to use KDD CUP99 as well, but this uses the other algorithm k-NN explained earlier in this thesis.

The thing that makes this paper stand out in a way is that it not only uses k-NN the algorithm, but the researchers also tries to optimize the results by using clustering to enhance the performance of the classifier. When looking at the results presented it does not improve the performance on all of the experiments that had been done. But in the test where it is done two cross validations of the data set, the results i actually better then without the clustering [32]. Two cross validations means that the data set is shuffled around 2 times. In all of the other experiments done in the paper the ones who run without the clustering has the best performance.

2.5.4 Paper 4:

The fourth paper is called "Real-time anomaly detection systems for Denial-of-Service attacks by weighted k-nearest-neighbor classifiers". It is written by Ming-Yang Su et.la. [33]. In this paper the researchers run two different tests using weighted k-nearest neighbours and just k-NN without the training of weighted variables of the different features [33].

The tests are done starting with one feature and then they add one and one until they have 35 features and for each of these they have calculated the accuracy. The tests showed a different result using trained weighted features and not trained features.

Test A as it is referred to in the paper, they used non trained weights they obtained lesser accuracy then when using trained weights as in test B. When using trained weighted features the results were much better with an accuracy of 97.4 % at max and 57 % at min, whilst non trained had a gap of max 78 % and min 64 %. So one could say that the trained was a bit more stable whilst the non trained had a lager gap, which makes sense since the weights would give an indicator of what is important and not so important.

Part II

The Project

Chapter 3

Approach

In this chapter the focus will be to make a plan of what going to be done in order to achieve an "answer" to the problem statement. The plan will be structured into steps or phases that is going to be solved one by one to achieve the goal. It will be explained what is going to be done in each step and how it is going to be done.

3.1 Objectives

The objectives of this thesis is based on the problem statement, which is as follows:

Comparing the two algorithms SVM and k-NN, to determine which of the algorithms that yields the better overall performance, when doing anomaly detection using machine learning techniques

And then the conditions that is set in the problem statement section 1.1, to be able to differ the two algorithms is: The parameters that is going to be used to determine which of the algorithms that has the best performance is:

- Classification performance
- Time consumption
- Resource consumption

The main goal of this thesis is to implement two different algorithms using a programming language called R. This is going to be done using a web-Graphical User Interface(GUI), that is a client side of the R server that is set up as a background software. This software is running on a server with many CPUs to be able to handle computations that is going to be done on the big data set. The algorithms both have several different packages that is available to use in R, so they will be installed and used to preform the predictions and training of the algorithms.

3.2 Experimental environment

The environment that is going to be used to do the experiments is a server with R studio running as a front-end application and R server as a back-end which handles the computations and the hard work. The front-end is where the experiments is going to be develop and where the results will come up at the end. The server is accessed using a browser and a specific URL. The website is running a R studio application. There is also the possibility of accessing the server using Secure Shell(SSH). The SSH is useful when measuring the CPU usage, timing of the experiments, etc.

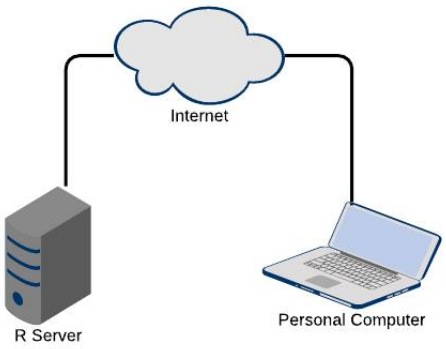


Figure 3.1: This is how the setup looks like for the experimental environment.

It is a very simple setup, it is just a server running a service and that can be connected to via a web-browser or a ssh connection. Bringing up a sketch like done here in figure 3.1 might not have been necessary, but it is the only setup that is being used and this illustrates the simplicity of the setup in the thesis. The important factor of the setup is that it has enough resources to run the processes needed to apply the algorithms.

Table 3.1: Physical Machine specifications

CPU:	Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz
Cores:	48
OS:	Ubuntu 12.04.5 LTS
Total Memory:	125 GB

As can be seen from table 3.1, the server that is being used to run the R server has a lot of CPU power due to it's 48 cores and each and every one of them have 2.30 GHz. Thus the computation or power needed to train and test the algorithm should not be a problem.

The Operating System(OS) running on the server is Ubuntu 12.04.5 LTS, which R server should work good together with and that should not be a problem since it is already running and has been used by other students to do even heavier calculations.

The server also has enough memory available, it has a total of 125 GB. Which is more then enough to run and do the processes that needs to be run, in order to do what is intended to do in this project.

3.3 Technologies

The technologies that is going to be used throughout the project is R server as mentioned above, and what is going to be done is to apply algorithms to a data set. This is planned to be done using machine learning techniques. The specific machine learning technique is supervised learning.

3.3.1 Supervised Learning

Supervised learning is as stated in the background chapter a machine learning technique that uses labeled data when it is trained. What this means and how this is going to be done practically is that when the algorithms are trained they are also trained with all of the facts known. So it has all of the features available and learns what the values in the different features can represent.

3.4 Chosen data set

The data set that is going to be used is the improved KDD CUP99 data set, which goes by the name NSL-KDD. NSL-KDD is as stated in the background chapter a improved KDD CUP99 data set. The new revised data set does no longer contain several of the problems the old one did. These problems were as mentioned in the background chapter inherent problems, lots of duplicated data and many other problems that could give

a false and very good result according to many researchers that has studied the data set thoroughly.

Features in the data set

The data set has a total of 42 different features whether it is the anomaly or the multiclass data set. This will in a real life scenario might have different weights, i.e. flags would be a much more important than many of the other features in the data set. For those who knows a little about network security a single SYN packet is not so important but if there is a whole TCP handshake this could be something that might be of interest. At least if the IP the machine is talking to is or has been related to something suspicious then this would be something of interest. So the weighting of the different features in the data set would in a real life scenario have weighted features that count more then some of the others.

The different features are both integers and characters, so in order to apply the algorithm to the data set the features with characters has to be converted into integers. When saying integers, all of the character values has to be changed into a numeric value. The numeric value can then later on be changed into the original value which is a string. This is what is going to be done to features like protocol type, service, flags and so on. Just to give even a clearer understanding of what is going to be done see table 3.2 below, which uses flags as an example of how this converting is going to be done.

Table 3.2: Changing categorical values into continuous values

Flags in characters	**Flags in Numeric**
SH	1
SF	2
S3	3
S2	4
S1	5
S0	6
RSTR	7
RSTOS0	8
RSTO	9
REJ	10
OTH	11

In table 3.2 all of the different values of the feature flags is listed and what value it will get when converted into numeric so that it can be used to train and predict the outcome of the different algorithms.

The data set consists of 42 features as mentioned, these are:

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

When reading all of the different features in the list above, it can be some who might say that this is not exactly a real output from a tcpdump or something like that since it has no source- and destination-ip-addresses, ports, etc. And that some of these features are not like what one might come across in a normal tcpdump or a network log file. In a network log file there would normally not be specified which of the services that each is connected, instead there might be a port number that indicates sort of the same thing. It is not exactly the same thing, and it might have been easier using ports, but that is the way the data set is set up and it is a very well known data set when working with machine learning.

Example of entries in the data set

For those who never have seen a data set like KDD or read a network traffic log table 3.3 will give a brief introduction to have the data set i structured. The features that is displayed is some of the key features there are also some other that is not in the data set. This is just the five first features and the last one. These are the once that makes sense to show in a table like this, many of the other features as noted in the list above does only contain ones and zeros. That would just take up a lot of unnecessary space, some of these different ones and zeros could be an important factor when applying the algorithm. But for the purpose of giving a visual presentation of the data set it would be more of a distraction then a positive help. This is why there is multiple punctuation marks between the feature source bytes and attack to indicate that there is also some features that is not displayed. If there is a need to see the whole data set it could be downloaded from the reference list [22, 28].

Table 3.3: Example of entries in the data set

Duration	**Protocol type**	**Service**	**Flag**	**Source bytes**	**.....**	**Attack**
0	tcp	ftp_data	SF	491	**.....**	Normal
0	tcp	private	S0	0	**.....**	Neptune

3.5 The plan

The initial plan for the project will probably be changed throughout the project, but to have a high level plan that should be followed is always a good thing. What is meant by saying a high level plan is that there should be some sort of plan that should be followed, the steps might be solved in a different order then what is listed and so on. The main thing with the overall plan is that this is what needs to be done in order for the project to be finished when the deadline day comes.

The high level plan for this project looks like what is displayed in figure 3.2 explained in a very simple form using a flowchart to explain the different steps in the project.

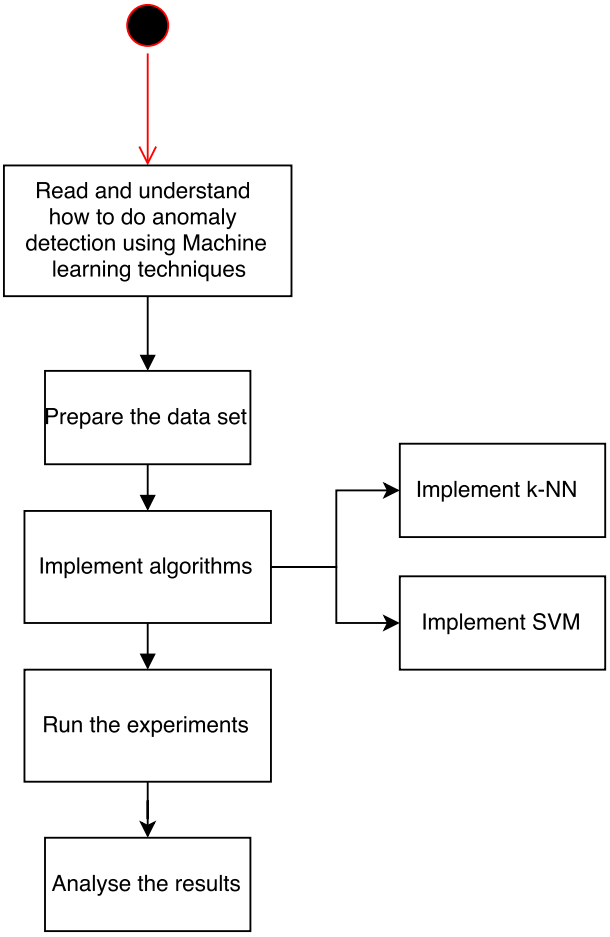


Figure 3.2: This is how the general plan looks like. It is explained in a very simple form just to make clear the steps of the project.

As can be seen from the UML diagram in figure 3.2, the key steps is very generalized and needs further explanation. A UML diagram is used or a flowchart might be a more correct notation of the diagram. What UML stands for is Unified Modeling Language, and is quite a normal way of presenting flows of a project or flows of a program etc. Figure 3.2 might not be exactly after the rules and standard of UML diagrams, as it is missing a ending point i.e. This is left out because it is not a set plan, it will probably change and there is more steps in between those steps listed. The purpose as said is not an exact plan, it just an outlining of what needs to be in place in order to have some results at the end of the project.

The first step listed is that the researcher needs to understand what machine learning is, how anomaly detection using machine learning techniques works and how it can be applied to a data set or a data stream. The background chapter will list some references that can be useful to getting a good understanding of machine learning and how it can be used to do anomaly detection.

The next step will then be to prepare the data set. The preparation needed is that the data set must be converted as explained earlier, the values containing characters must be converted in order for the data to be processed by the algorithms. Also the features or columns that has zero values needs to be edited or removed depending on the columns importance. Some features do not have much effect on the decision making, due to having low variance. There are multiple columns that contains only zeros and ones as said earlier, some of these are not as important as some the others looking from a real security perspective. For the data set to be useful these things has to be done, otherwise the program that is going to written will have problems running. Often the features that contain characters are referred to as categorical value, and the numeric are often referred to as continuous. Which may not have to explained to everyone, but what this means is that the continuous are values that do not break or interrupt.

After all of the data points in the data set has been made continuous, it is time to start implementing the different algorithms. The plan is first to start with k-NN and then move on to implementing SVM. k-NN is going to be implemented using a package that can be installed on R studio or R server. To install a package that is available from R servers is not a difficult job. This is done using this line in the R console:

How to install packages in R:

```
1 install.packages("Package that is going to be installed")
```

The packages that is going to be used to do the actual implementing of the algorithms k-NN and SVM are two different packages. As said these needs to be installed, the package that is going to be used for k-NN is named Class. In the package Class, k-NN is one of the features that are made available by the package. For implementing SVM the package called e1071 is going to be used, it contains different features that can be used when working with the algorithm SVM.

The developing will as stated earlier be done in R studio. There will be written scripts that implements the different algorithms and that does all of the formatting needed for the algorithms to be run. One important thing that needs to be done is to add another column to the data set. The row that will be added is going to contain type of attack the attacks that already is in the data set belongs to. As explained about the data set earlier, the attacks are listed but not the group or type of attack it belongs to. Say that i.e. neptune is a DoS attack, nmap is Probe and so on. The row containing the attacks will have to be looped through and the next column will contain the group that the attack belong to.

The next step is running the experiments. There is at least going to be run four experiments. These will be more explained in section 3.6 called Experiments. All of the different algorithms can be considered as an experiment, and they might be listed as experiments each and every one of them depending on the different results, accuracy, etc. The thought is that they should be "experiments" but not listed as experiments, the results from the algorithms will be part of the experiments. This will all be in much greater detail be explained under section 3.6 and in the results.

When the experiments are finished and the results have been presented, what will then be done is to analyze the results. The analysis will be an important part where the results are listed and explained. In the discussion it will be discussed how the project went, what was the difficulties with getting the results, is the result satisfying, etc.

3.6 Planned experiments

The planned experiments is the ones that is thought out to be used to differ the two algorithms, when it comes to performance, accuracy, time usage, resource usage etc. There will be done different experiments depending on what the focus of the experiments is supposed to be. The tests or experiments should be performed so that the tests could be representative and the algorithms should have the equal testing environments, resources available, etc. If these standard rules for tests experiments are followed the results from the experiments might be considered as valid data. Valid data is that the data presented could be defined as trustworthy and reliable.

3.6.1 Experiment one: Binary classification

The first experiment that is planned is to look into the two different algorithms. This is going to be done using a binary version of the data set. This version contains just two different "attacks". Attacks is put in quote marks because there is actually just one of the values in the columns of attacks, that indicates that it is an attack, the other is normal. So the attack column does only contain the values:

- Normal
- Anomaly

This should make it easier for the algorithm to process the data set and it should give a better result at the end, due to the fact that there is only two different groups it could be classified as. It should also run faster since it only has to work with two classes.

3.6.2 Experiment two: Multiclass classification

The second experiment will be done using multiclass classification. Multiclass classification is when the outcome of the prediction done by the algorithms can be more then just two classes. In this case the outcome can be five different classes. There is only going to be used the groups of attacks or types of attack, not the actual attacks. This is due to that it would take a very long time and require a lot of resources to implement when using all 22 different attacks. So therefore it has been decided that using the group of attacks which are:

- Normal
- Probe
- Dos
- R2L
- U2L

So the algorithms will in this experiment try to predict which of these classes the flows of network traffic being fed into the algorithm belongs to. Then it will be compared with the original data set to see how good of an performance or accuracy it has reached. It will use some of the data set for training the algorithm and the rest for testing. It is considered a thumb of rule to use about 60 % of the data set for training and the remaining 40 % for testing and validation. In both this experiment and in experiment one there is going to be done cross validation on the data set.

What cross validation means is that the rows of data is switched around. This is done so that the data set will almost never have the same order so that the algorithm cannot use the order of the data that is being fed in as a factor in the prediction. The cross validation should be done before both

of the algorithms are run, and the number of cross validations is not a set rule. In this project there will be used 10 cross validation in order to ensure that the data set is mixed up. A good example to compare cross validation and data sets to is a deck of cards, it is important to shuffle the deck before it is being used in order to keep the game fair.

3.6.3 Experiment three: Time consumption

Time is a very important thing today, everything should run fast and use less time then what is already available in the market today. For the algorithms and the idea about using machine learning techniques to do anomaly detection instead of using a normal IDS, the need for it to be fast is a very important factor. It should be able to process as much rows or lines from the data set or data stream(which is what would be the input in real life scenario where machine learning could replace the use of IDSs) as possible.

From the papers already read, time might be one of the bottlenecks, due to it uses some time to process, train and learn the difference between an attack and normal legit traffic. The experiments will then be measured from start until the end of the experiments. What will be considered fast and slow might be difficult to determine, due to there is not much to compare with other then research already done with the exact same algorithms and packages used. It could also be compared to solutions used today such as IDSs and so on, but they work in a totally different matter. So the plan is to measure both the experiments when the classifications is run and compare the results of the two algorithms. This is going to be done when testing both with binary and multiclass classification. The binary should as said in the experiment description run faster in theory, but if this is the case is going to be tested.

The problem with comparing time consumption of an normal IDS and an algorithm is that as said earlier in this paper that an IDS often searches for a specific string, IP or some set value. This set value is often tied to a known attack using IPs, strings in a payload and so on. This pattern recognition is very fast and can alert or block the traffic immediately, whilst a algorithm needs much data just for training and it takes time to collect and apply the data. The advantages with machine learning is that it can learn from training and it will work fast once it knows what patterns to look for.

The concepts of the two are very different and many will say that comparing them will be impossible. When comparing the time aspect would not be the main focus, the center of attention would be the detection rate or classification performance. Time is of course an important factor, but if it takes just a little more time to use machine learning and this gives a much greater detection rate it might be able to take over for the traditional IDS/IPS. But on the other hand if the results are bad on both detection and

time this might not be the case.

3.6.4 Experiment four: Resource usage

The fourth experiment that is planned to take place is to measure how much resources are being used. When doing heavy computation the CPU usage is an important factor. The CPU usage will be stored while doing the different experiments to see if one of the algorithms will use more then the other, how much they use, etc. It will also look at memory usage and other resources it might take advantage of. This will be done tracking the process of R server. When one logs into the GUI of R studio, R server starts a session belonging to the user that logged in to the server. This will start and the tracking will be done using this session. It is the process that is effected when running anything in R studio.

When this test will be done, there will be no other users that uses that exact server other then the process of running the algorithm and standard processes that is run on the server. This is important so that the results of the tests will not be altered by any other processes eating resource, that the server can provide. It is a common practice when measuring usage of resources, that all unnecessary processes and other factors that can influence the end result is turned off.

The results from the experiment will be taken into consideration when comparing the two algorithms. The one that uses less resources and if the accuracy is good on the same algorithm can be heavy weighted. A "cheap" program that is effective, precise, fast and uses least resources as possible is the most optimal solution. If they both uses the same amount of resources this experiment will not have to much to say unless one uses more or less than the other. Then the resource usage will not have to much to say on the end result.

3.6.5 The results of the experiments

The results of the experiments is what is going to determine which of the two algorithms that has the best results overall. It will determine which will be announced as the most effective and best algorithm, when it comes to doing anomaly detection using machine learning techniques. At least in this paper, if the numbers and the tests will be considered to be valid and considered as good research is up to other researchers to decide and determine. The results will be presented in the chapter Results II and will be analyzed in the analysis chapter and the discussion chapter.

3.7 Constraints of the project

There are some constrains to the project due to time and restrictions when it comes to user privileges on the server and some other small things that can affect the project.

3.7.1 The setup

The setup has some restrictions when it comes to privileges as mentioned. The author do not have root privileges on the server, but the supervisor does. The supervisor is able to do some alterations to the setup, like configuring the R server and so on. One of the initial thoughts was to use a GPU server to speed up the implementation of the algorithms and make the training and learning process go faster. When starting reading about this the R server do have some support for this but there has to be used a specific type of servers, and it needs to run proprietary software. So this thought was discarded, but the server had enough CPU power and other needed resources. There is also some operations that might not be able to do due to the need of root user privileges, like monitoring the resource usage might be difficult without root user. But this will have to be dealt with along the way.

3.7.2 The time

The time could be a constraint due the fact that this is a lot to learn in the period of time that is available in a master thesis. The author has little prior knowledge of doing what needs to be implemented. So it might be time consuming to read up on what needs to be read and so on. But a challenge is good and a lot could be learned from the project.

3.7.3 The technology

R server or R studio which is mainly going to be used is a very common and known program to do statistical calculations and computations in. The language R is not as logical to someone who has learned any other programming language. There is a saying that once you have learned a programming language it is easy to learn any other language. This due to there are so many websites and it is easy to search if you are stuck with some errors. From experience this may not be the case when working with R. There are some forums that one can turn to with questions, but the answers or solutions when one is stuck may not be as easy to search for as many other languages. But the plan is to use R as said an implement the algorithms as good as possible.

3.8 Alternative approach

There is an alternative approach that could have been used instead of the suggested approach in this chapter. What could have been planned to be done different is that there is an option to use Python to implement the algorithms. This option would also have been a good option, Python as a programming language might not be the fastest of all programming languages, but neither is R. Python does also have multiple packages available to implement different algorithms for classification and machine learning. The reason for choosing to go with R was that the supervisor had

prior knowledge to implementing algorithms to do anomaly detection in R. There are also some other reasons like, when using R it easy to make graphs, plot the results and gives a good overview of the data set when read in as a data frame. There is also many other neat functions that is built in when using R, such as getting rid of NA values and other small functions that is good when working with data sets.

R was the chosen technology due it being a good way of implementing the algorithms and having a lot of built in functions that could help along the way. It has a very robust software that could be altered and configured easily to what needs to be done. Installing the different packages that is mentioned easily done straight into the console that comes with the web GUI.

3.9 Other implementations

This section will describe other important implementations that are not mentioned earlier, but needs to be done in order, so that it is even possible to do the experiments at all. There are some key implementations that still has not been described much, and these will be described and how this is going to be done.

3.9.1 The scripts

There is going to be developed four scripts that will automate the process of doing the classification. This will be done with both of the algorithms and using both binary and multiclass classification. So there will be written two scripts for using k-NN and SVM with binary classification and two where the same algorithms will be implemented using multiclass classification.

Scripts using a binary data set

The script that is going to written for doing anomaly detection using a binary data set will be a pretty simple implementation. The data set will have to be prepared and this is going to be done by the script. That is the first thing that needs to be done in order for the algorithms to work. Then the next step will be to remove the features in the data set that contains zero values or has NA values, this is explained earlier in the paper. Then the algorithm can be programmed. These steps are the same for both k-NN and SVm in the binary data set. When using the binary data set, the classification will have a false/positive scenario. Where the false positive is translated into anomaly or normal.

Scripts using a multiclass data set

The script for using multiclass classification will be written in the same matter as the one using binary. There will have to be some modifications to the data set, such as zero values and NAs etc. But instead of having

just to different classifications in the end results, there will be five different classifications. This is due to it being five different types of attack in the data set column called type_attack which is the one that is going to be predicted.

3.10 Expected results

What kind of results to expect is a difficult to assume, but it could be based on the papers read and the results found in these. In the papers that is referenced in the background chapter one could assume that the results could be very good when looking at the aspect of accuracy. When saying accuracy what is meant is the classification accuracy, how good the algorithms can predict. Both k-NN and SVM are known to have good results, when looking at accuracy, but one issue that is mentioned in some papers is that it could be time consuming to get these good results.

Chapter 4

Results I: The design and Implementation

4.1 Overview

In this chapter the implementation of the algorithms will be described more than in the approach and the design part of the approach. It will only contain information on how the design became and how this was done. There will be used some diagrams, tables and figures to explain flows and some logical examples. The will also be used some code examples that is of importance for the implementation of the design. The design will be explained and tried to made as simple as possible so the reader could easily understand.

In the approach there is only a suggested plan of what was going to be done, and in this chapter it will be explained what actually happened. There has been some small changes made throughout the project, these changes were made to enhance the project. In this chapter it will be explained what has been done and how it became so. All of the different plans mentioned in the approach will be discussed and there will be an thorough explanation both visually and in text. To use the word design might be a bit of an overstatement, since there is only certain ways it is possible to implement the algorithms. There is some sort of standards that the data set must follow or be optimized for. So this standard will be utilized when design the project, including the implementation and the scripts that are made. When using the word design in this thesis it will be about how the experiments(scripts) were made, how they came to be the way the became and why it was necessary to do it in the way it has been done.

4.2 The environment and data set

The environment or the setup as it is listed in the approach chapter ended up as planned. There is a server serving a R studio GUI, with R server running as a back-end. The R studio can be accessed from any browser from any operating system, and this is where the actual coding and importing of the data set happens. One reads in the data set from a .csv, .txt or .arff file which also is the case when using the data set that is used. A .csv and a .arff file looks very much a like and both can be read in by R studio. The only difference between an .arff file and an .csv file is that the .arff in this case also contains the different attribute names, or information that can be read in together with the data set in the same operation. Attributes names is the name of the features listed in the approach. When using the .txt file that comes with the NSL-KDD one have to manually set the column names (or feature names). This is going to be explained how it will be done in the section preparation of the data set.

The data set is also the same as planned. It was first the thought to use the original KDD CUP99 data set, but since this has been criticized heavily for its inherent problems and several other issues the NSL-KDD data set was chosen. This data set should be an improvement on the original data set and is as big as the original data set. So the NSL-KDD should not be a negative change but a positive enhancement of the project, and should give a more realistic end result then the original due it being more like a real network log. There are still being used the same features in the data set, so saying that it is just like a network log would be incorrect. It still contain features like service instead of ports, which probably would not be the case in a normal network log. The author of the NSL-KDD has done a thorough investigation of the original data set and tried to eliminate the findings of weaknesses, redundant data etc. It has been acknowledge as an improvement by other researchers that has done a comparison of the two.

The supervisor of the thesis is the one that has root or superuser privileges and can do modifications to the server and configure the R server on request and this has been an okay solution. The setup is as said the same as in the general plan made in the approach. There has been no trouble with connections and the server has been up and running throughout the project. The only modification that is done is to install some software to monitor the CPU usage. This was done in order to measure the resource usage, which is the fourth experiment. So the privileges on the turned out not to be a problem after all, there was no problem with doing any of the different experiments. At least when it comes to the need of superuser or root privileges.

4.3 The algorithms

The algorithms that has been implemented is the ones that is listed in the approach and that is explained in the background chapter. These were SVM and k-NN. They are the most used classification algorithms and that was one of the many reasons for choosing these two.

4.3.1 K-Nearest-Neighbours(k-NN)

k-NN is an algorithm that is widely used when doing classification due its simplicity in the way the algorithm works and in the way of implementing it. As long as all of the values is continuous and the NA values are removed or changed into values. This goes for most of the algorithms using numeric calculations to do the predictions, there is also some that uses text for pattern recognition. But in both of the algorithms that is being used in this project there is factors or numeric values that should be in the data set for the algorithm to work properly. When using an algorithm there is often or in most cases a need to give the actual model that is going to made one or more parameters. The parameter that is used when working with k-NN is the *k*. It is the one parameter that is giving to the algorithm in order for it to know how many of the nearest neighbours it should let be a part of the majority vote. There is many researchers that have different suggestions on how to determine what the *k* should be. Some say that they follow the standard that has been suggested by the research done by R. Duda et. al. [34]. This standard says that the thumb of rule when working with the algorithm k-NN, the *k* should be as near the square root of the numbers of entries in the data set used. Others just optimize the *k* to get the best result as possible, often the lower the value of *k* is the better are the results. So there is no actual standard or way of calculating the *k*, when looking at what other researchers have done previously. There has been attempted to use both the thumb of rule and the optimization method by try and fail.

4.3.2 Support Vector Machine(SVM)

SVM is also a very widely used algorithm when doing classification, this is mainly due to it is a good accuracy and that it can be used to do multicalss classification. There are lots of papers written about anomaly detection using machine learning, where SVM is the algorithm used as the classification algorithm. Many of the papers can show accuracy in the numbers close to a 100 % which would be the wanted outcome, but is almost impossible to achieve at least in a real life scenario. In a real life scenario, where the attacks are so many and the threats just keeps developing and getting harder to detect. It would require a lot of training data in order for the machine learning could replace the normal IDS. Just even talking about numbers that many of the researchers have achieved in their experiments is a high achievement and is a big step for machine learning. In most of the papers listed in the background chapter the

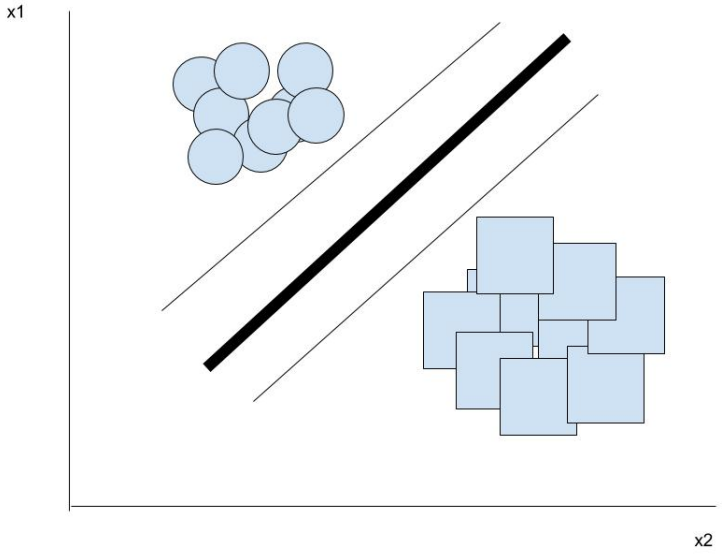
researchers have gotten results as high as 97 %, which is a very good result.

SVM can be used in many ways and there are several parameters that is given to the model. One can use SVM using a linear kernel, what this means explained in a very simple matter is that the hyper planes(which is explained in the background chapter, both visually and in theory) are linear. There is also some other kernels that can be used like radial, polynomial etc. In this paper the linear will be used, that means that it will have linear supports vectors as explained and visualised in the background chapter in figure 2.2. It gives a pleasing result and is what is normal to use in the examples and papers found that uses the algorithm SVM.

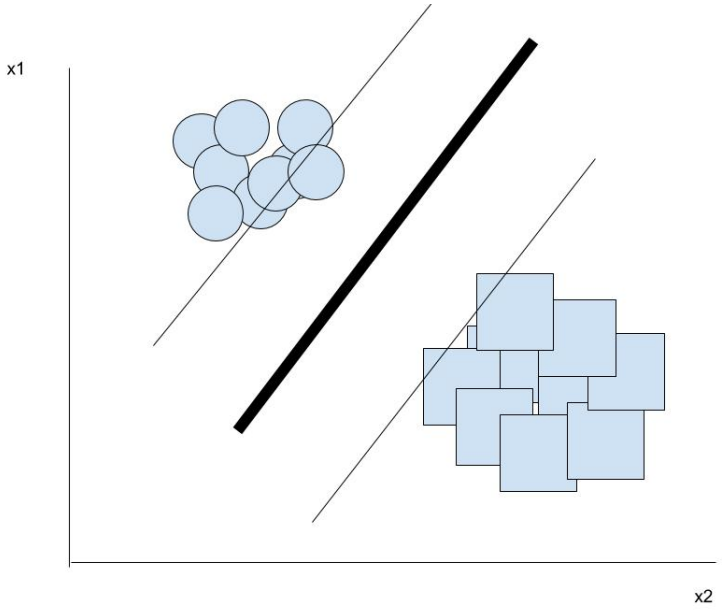
The SVM algorithm is also fed with some other parameters that can be crucial when it comes to the accuracy of the classification. These are *gamma* and *cost*, which is to parameters that the calculations of the model that SVM uses to create the predictions. These two parameters have to be optimized before the classification process can be done. The gamma and cost parameters have different functions. The gamma parameter or γ which is the mathematical representation of it is used to define how much influence on training example has on the model that is being created. Low values will mean that is does not have too much of en effect each and every one of the training examples. Whilst a high value means that every training example has a high effect. What can be drawn from this is that the gamma can be very crucial and needs to be optimized.

The cost or C is the cost of misclassification on the training data. It is sort of a penalty if there is misclassification in the training data, and if the cost has a high value this will make it a more strict classification and there will be smaller margins of error. Whilst if the cost value is low there will be room for more errors and the classification will be more "loose" or not so strict. The optimal is to get a cost that leaves room for some errors but not to many so that it will learn. When having a high cost value there will be a much more strict classification of the training data and the aim is to classify all the training data correctly.

Both C and γ can be optimized by doing a so called grid search, this will be explained more about later when the actual implementation will be explained. The importance of getting these two parameters correct can be crucial as said, if i.e. C is to big it could lead to over-fitting the model. Over-fitting the model means that the hyperplanes will not be placed correct and it could lead to unnecessary incorrect classification. The hyper planes should be placed in the optimal place between the different classes and for this to happen the cost and gamma needs to be correct. Otherwise the hyperplanes will be placed incorrect and can cause a unfortunate classification result. As can be seen from the figure 4.1. The figure marked with an *a* is what an optimized and fit model would look like, whilst figure *b* is an over-fitted model where gamma and cost have been miscalculated.



(a) Example of how a hyperplane should look like when fitted correct.



(b) Example of how a hyperplane can look like if over-fitted.

Figure 4.1: Figure a and b is example of how a satisfying and a unsatisfying hyperplane

4.4 Preparation of the data set

As already explained in the approach there are some certain things that has to be done before the data set is ready to be used by the algorithms. There are multiple conversions and modifications that needs to be done, in order for the algorithms to run properly and without error due to the data set.

4.4.1 Read data set with names on features

The first thing that was done with the data set, was read it into R studio in order to see the whole data set and to use it afterwards. This is easily done by this line:

Reading the data set

```
kdd_train=read.csv(file="filename", sep = ",")
```

What this line of code does is that it reads that data set and stores is as kdd_train. When reading the data set into a data frame as it is called in R, it splits the data into columns or features using "," as a separator.

The next step is to put labels on the different columns created when reading the data set. This is done by reading a file containing all the feature names and applies it to the data set, so that the rows has the correct names and is easy to call upon when using and modifying the different features. This is not to hard to do but it is important to get a neat and good data set to work with. This is how it was done in the script that is used in this project:

How to read the feature names and applying it to the data set

```
# reads the names of the columns
colnames <- read.table("names", skip = 1, sep = ":")
# Sets the names on the trainingset
names(kdd_train) <- colnames$V1
```

4.4.2 Categorical values vs. continuous values

One important thing needed to be done before the algorithms can be applied and run is the preparation of the data set. It needs to be done as the data set contains categorical values such as protocol_type, service and flags. These are categorical values which means that they are not continuous, something they have to be for the algorithm to work. What is meant by categorical values is values in the column or feature, is a value that is not numeric, factors or vectors which are continuous values. There is packages that can be used to convert these categorical values into numeric like the package *model*. The package model has a function that can read in all the different values and create a numeric value for the categorical. The conversion could also be done manually, by using a loop and if statements, or a replacing in the data set like shown in the table in the approach chapter. In this chapter the code for actually converting the values will be shown and explained briefly as can be seen in the code block underneath.

How the converting of the categorical values is done

```
1  kdd_train$flag = as.character(kdd_train$flag)
2  kdd_train$flag[kdd_train$flag == "SH"] = 1
3  kdd_train$flag[kdd_train$flag == "SF"] = 2
4  kdd_train$flag[kdd_train$flag == "S3"] = 3
5  kdd_train$flag[kdd_train$flag == "S2"] = 4
6  kdd_train$flag[kdd_train$flag == "S1"] = 5
7  kdd_train$flag[kdd_train$flag == "S0"] = 6
8  kdd_train$flag[kdd_train$flag == "RSTR"] = 7
9  kdd_train$flag[kdd_train$flag == "RSTOS0"] = 8
10 kdd_train$flag[kdd_train$flag == "RSTO"] = 9
11 kdd_train$flag[kdd_train$flag == "REJ"] = 10
12 kdd_train$flag[kdd_train$flag == "OTH"] = 11
13 kdd_train$flag = as.factor(kdd_train$flag)
```

kdd_train is the data set in its original form with names on the features. The first line of code on the code block is used in order to assure that the column or feature are characters. This needs to be done so that when searching for the different types of flags and replacing them, what is done in line 2-12 is successful. Then in line 13 it is converted back to be a factor, a factor is a numeric representation and it needs to be numeric as stated earlier in the project. The example used here shows how all of the different types of flags that is in the data set and how it is converted into numeric values. Line number 2 in the code block shows how "SH" is converted into a numeric value, here represented with the number "1". This is done to all of the categorical values in the data set, these are as mentioned over protocol_type, service and flags.

4.4.3 Partitioning the data set

When doing anomaly detection using machine learning techniques the data has to be divided into at least two data sets. This is in order to have a training data set and at least a testing data set. At least in this context is used since there is very often divided into three data sets as well. The reason for dividing a data set into three data sets is since there should be a training data set, a testing data set and a data set for validation. Which is a good thought, but in this project the validation will be done using cross validation as mentioned in the approach. So having a part of the data set for validation will not be necessary. Instead the part of the data set that used for validation, when not having cross validation(CV), will be added to the testing set. The standard of the partitioning data sets for training and testing when working with classification algorithms is not set in stone, some uses 80:20, 75:25, 70:30 and some uses 60:40(Training-data:Testing-data). These are all percentages of the data sets used for training and for testing. The words validation and testing is also mixed up a lot in different papers found, when the word testing the algorithm is used in this paper it refers to the actual predictions that is presented in the results. Training set is the data set that the algorithm will use to learn the patterns from and should after the training data has

been applied be able to recognize the different attacks in the test data set. How this partitioning is done in R studio is by using a package called *caret*, it contains a function that does the partitioning so to say for you.

How to do the partitioning in R using the caret package

```
1 trainIndex<-createDataPartition(kdd_train$type_attack,p=.6,list = F)
2 kddTraining = kdd_train[trainIndex,]
3 kddTesting = kdd_train[-trainIndex,]
```

As can be seen from the code above, the *caret* package contains a function called *createDataPartition*. This is the function that divides the data set into two different data sets. In this project it is used to create a training and a testing data set. The partition of the data set is stored as trainIndex, and it is two data sets, where the first partition is on 60 %. This can be drawn from the parameter in line 1 *p=.6*. In line 2 and line 3 the *trainIndex* and *-trainIndex* is the two values of *trainIndex*, which are the 60 % data set(*trainIndex*) and the 40 % data set(*-trainIndex*). These are in line 2 and line 3 stored into values called *kddTraining* and *kddTesting* in order to make it easier to use and differ the two data sets.

After all of this is done the data sets is almost ready to be used. There are some modifications that still needs to be made before it is ready to be used by the algorithms. There are several steps before the actual calculation can be made as explained in this section of the chapter. It is not just to use any data set at hand and do the predicting of the algorithms. There is several steps and modifications that has to be in place just for the algorithms to be applied. The cost and gamma, are two values that has to be optimized after all the modifications is done. There is a lot to be done and to remember when working with classification algorithms. It is enough to grasp and if one of the steps above has not been preformed there will be errors. Of course there are many data sets that is already optimized and modified, so that one can just go ahead and do the implementation of the algorithms. A good example of a data set like that is the one that comes with R studio called *iris*. It is a data set that just needs to be split up into training and test set. *Iris* is very often used by how to's and other websites explaining how to use the different classification algorithms and when troubleshooting. The *iris* data set contains data based on flowers and the only categorical value in the data set is the one that says which flower it is. All the other values is numeric.

4.4.4 Zero values and NA values

The next important step in order to make the algorithm code run is to remove the NA and zero values. Zero values and NA values were explained earlier in the thesis in detail, but just to give a very short description it will be shortly explained here as well. Zero values is a notation on the columns that does not contain anything other than zeros.

There are literally no values in the column so that it will not have an effect and makes it difficult for the algorithm to be run. There can of course be zeros in the data set, but if there are only zeros in the whole column this does not give any helpful information. Then there is also the NAs, NANs and INF values. These will either have to be converted into values or removed. What the NA means is that there are some values missing in the data set. Like shown in table 4.1, where it is displayed that there is a value which are not set in the column Source bytes. This can be due to many things, it could be a missing value in the original data set, or there can have happened something when converting the categorical values etc. NAN means that there are some values in the data set that is not numbers.

Table 4.1: NA values

Duration	**Protocol type**	**Service**	**Flag**	**Source bytes**	**.....**	**Attack**
0	1	10	3	491	**.....**	Normal
0	1	11	8	NA	**.....**	Neptune

That there i.e. is values in the data set that are characters or a value that is not numeric. What INF means is that the value of the data point that is being marked as INF can mean that the value returns negative value when used for example dividing by zero is a example. It could also mean that when taking the logarithm of the value gives a negative result. This values will either have to be removed or converted as said. This could be done by looking for the different values in the data set, and could be done in many variations. In the NSL-KDD after having modified, converted and split up into two data set there were some NA values. These were removed by using the code listed underneath.

How to do get rid of the NA values in the NSL-KDD data set

```
zeroVarianceFeatures <- sapply(kddTraining, function(i){
  if((is.numeric(i) & !any(is.nan(i)) & sd(i) >0) | is.factor(i)
  | is.character(i)) TRUE
  else FALSE
})

sapply(kddTraining, function(x)all(is.na(x)))
naValuesTest <-  function (x) {
  w <- sapply(x, function(x)all(is.na(x)))
  if (any(w)) {
    stop(paste("All NA values are found in columns", paste(which(w),
    collapse=", ")))
  }
}

naValuesTest(kddTraining)
```

What this block of code does is that it looks for NA, NAN and zero variance. If there is any zero variance these are removed from the data set. That is what the zeroVarianceFeatures does, it looks for zero variance in

the data set and prints true if there are any. The next function that returns all of the columns containing any NA values and writes them out to the console if there are any found. These NA values will either have to be removed or converted into values that can be used instead. There is the possibility of just creating some random numbers instead for the missing values, but this could interfere with what trends or patterns the algorithm recognizes. In the case of this project there are so many features in the data set so removing some the columns that has NA values might not be a problem. In the project the features or columns containing zero values and NA were removed from the data set. This columns were all features that could be left out of the data set, since they are not much of value in a security aspect.

There is also a risk when removing features like done in this project. Features that one might think has no effect when applying the algorithms, maybe more important to recognizing a pattern then one might believe in the first case. If the feature i.e. is *dst_host_serror_rate*, this does not give much info when looking at it with a security aspect. It says the rate of destination error rate. If there has been any errors on destination is not an important feature, at least when comparing it to flags, protocols, bytes, service etc.

4.5 The experiments

There has been made 4 different but somewhat similar scripts in R during the project. These are meant for experiment 1 and experiment 2 as stated in the approach and also mentioned earlier in this chapter. These script should automate the process of running the algorithms and optimize them when possible. All of the different scripts has been tested several times and should give a result on each of them. All of the scripts can be found in the appendix and could also be downloaded from Github. To remake or reuse these scripts is possible, there is only one small obstacle. The NSL-KDD is not open to the public, but the maker of the NSL-KDD data set distributes it to organizations, students, etc. An email to the author of the NSL-KDD, listed on the website, stating what it is going to be used for and by whom should be sufficient [28]. The scripts that are made uses the NSL-KDD data set and is therefore optimized for this data set only. They could be modified to take another data set as input and optimizing this, but then there would have to made some modification.

4.5.1 Script one

The first script that was actually made was the one implementing the algorithm k-NN, using multiclass classification. This was done because implementing the binary classification would be very easy, if the multiclass was already working. These two different data sets used for binary and multiclass is equal, the only difference is the last column as said earilier several times. The multiclass needs more to be done in order for it to work due to having five classification options instead of just two as the binary has. There is also the problem with converting the different attacks into the groups or types of attacks, e.g neptune into DoS etc. This has to be done for the SVM and k-NN multiclass classification implementations. This was then the first thing that was attempted after all the other modifications, splitting of the data set etc. was done. What is done in order to create a new row(43) in the data set, is to loop through the whole data sets column 42 which is the column with the feature attacks. In this loop there are if and else-if tests that looks for the different attacks and if one is found it groups the attack it by adding the attacks group or type in the column right beside of it. This will then create a new last row containing the different attack types or groups. How this done practically in the script is listed in the code block named Creating column of attack types.

Creating column of attack types

```
kdd_train$type_attack <- 0
# loops through and writes the correct class based on
# the subclass which is attacks
for(i in 1:nrow(kdd_train))
  if((kdd_train[i,42]=="smurf")|(kdd_train[i,42]=="neptune")|
  (kdd_train[i,42]=="back")|(kdd_train[i,42]=="teardrop")|
  (kdd_train[i,42]=="pod")|(kdd_train[i,42]=="land")){
         kdd_train[i,43]="DoS"
  }else if(kdd_train[i,42]=='normal'){
         kdd_train[i,43]="Normal"
  }else if((kdd_train[i,42]=="buffer_overflow")|
  (kdd_train[i,42]=="loadmodule")|(kdd_train[i,42]=="perl")|
  (kdd_train[i,42]=="rootkit")){
         kdd_train[i,43]="U2R"
  }else if( (kdd_train[i,42]=="ftp_write")|
  (kdd_train[i,42]=="guess_passwd")|
  (kdd_train[i,42]=="multihop")|(kdd_train[i,42]=="phf")|
  (kdd_train[i,42]=="imap")|(kdd_train[i,42]=="spy")|
  (kdd_train[i,42]=="warezclient")|(kdd_train[i,42]=="warezmaster")){
         kdd_train[i,43]="R2L"
  }else if((kdd_train[i,42]=="ipsweep")|(kdd_train[i,42]=="nmap")|
  (kdd_train[i,42]=="portsweep")|(kdd_train[i,42]=="satan")){
         kdd_train[i,43]="Probe"
  }
```

The first thing listed in the code block is to create a new column in the data set called *type_attack* and it contains nothing. This due to the arrow and hyphen meaning that it should create a column or feature in the data

set. The symbol <- is the same as setting an equal sign, but the arrow is used very often in R. The data set is empty since it is set to zero or NULL, both could be used and gives the same result when run. Then the loop is set to go through all the numbers of rows in the data set, this is done so that it will loop through the whole column attacks. Then it looks for the different attacks and uses the number of the row in the data set the attack is in and store the group of the attack in the next column using the row number and the column number. This will also have to be done on the script for multiclass SVM as well.

The first script is put together by combining all of the preparation needed and the specific necessities for doing multiclass classification using k-NN. The whole script can be found in the appendix. The code that is executed to be able to use k-NN with mutliclass classification is listed in the code box here:

4.5.2 Script two

When the first script was developed and worked, the next implementation was then k-NN with binary. The binary data set should be processed faster because it only has two classes to classify the data as(Anomaly and normal as stated in previous chapters). The multiclass classification lead the way for the binary script, the scripts do not differ much. This binary classification script do not need the multiclass column, and there are some other lines of code that were unnecessary in this script. The difference can be seen in the appendix, by comparing the two scripts. As expected this script was not to hard to implement, when having the multiclass classification script as a starting point. It was simply to change the data set into the binary one and then remove some lines that is specifically needed for the multiclass classification. There was also done some small alterations to the script, like changing some pointers and so on. When saying pointers, what is meant by this is actually what columns it should look like and so on. The k-NN line of code displayed in the previous section is the same in this script. The k value is even equal, due to the data sets having the same lengths and the same amount of columns in the data set.

4.5.3 Script three

The script implementing the algorithm called SVM with multiclass classification, was the third script made in the project. There is also many similarities between the two scripts already made in this script. This is mostly due to the preparation of the data set, which is sort of a global configuration, in the sense that there are many of the same things that are done in both of them. But there are also some very important differences in the script that is specific for each of the algorithms. Like for instance the *tune.svm()* which is used to create a grid search for the most optimal cost and gamma values. These are two parameters that SVM uses as explained earlier. If none of the parameters are specified it will just use

the standard, which is set to 1 in both cases. This can give good results, but it is not optimized to be the best results possible using the algorithm. What *tune.svm()* actually does is that it uses the data set to compute a the best value out of a range that is set. This range can be big, but the bigger it is, the longer it will take to compute the optimized values of gamma and cost. There is also done cross validation in the *tune.svm()* so the data will be shuffled. This is also done in the actual running of the algorithm or the model made, might be more correct. In *tune.svm()* it uses a cross validation of three, what the value three means is that shuffles and mixes up data set three times. In the modeling of the SVM, the same procedure is done, but in that case it does ten cross validations. To use a cross validation equal to ten, will mean that the data set will be shuffled around ten times. How to compute or optimize the gamma and cost is displayed in the code box named How the gamma and cost is calculated.

How the gamma and cost is calculated.

```
1 tuneOutSVM <- tune.svm(as.factor(type_attack)~., data=kddTraining,
2         gamma = 2^c(-8,-4,0,4), cost = 2^c(-8,-4,-2,0),
3         tunecontrol = tune.control(cross = 3, sampling = "cross"))
4 plot(tuneOutSVM, transform.x = log2, transform.y = log2)
```

When looking at the first line in the script, which actually is the three first lines, but due to space limitations it has been split up into three lines. The *tune.svm()* or tuneOutSVM as it is stored as, contains ranges of gamma and cost that it tunes to check for the best value within that range. This range is equal on both cost and gamma and starts just over zero and ranges up to 16. These values is then tested to check which will give the best performance using the training data set. Line number four in the code box makes a plot of the support vectors using the log of the different values in tuneOutSVM. This is done to create a the lines and these is called support vectors or the hyperplanes mentioned earlier. These will be used to classify and differ the different classes in the result. After these have been optimized it is made a classifier. The classifier is made using the library *1071* and the function called *svm*. This function creates a model that can be used to predict the outcome of a given input or data set after it has been declared. How the model used in this project is made can be seen in the code block named How to create the model for SVM.

How the gamma and cost is calculated.

```
1 svmClassifier=svm(as.factor(kddTraining$type_attack)~ .,
2   data=kddTraining,core="libsvm",
3   kernel="linear",cross=10,
4   gamma = tuneOutSVM$best.parameters$gamma,
5   cost = tuneOutSVM$best.parameters$cost, probability=TRUE)
```

Here it can be seen that the kernel as mentioned earlier is set to be linear at the third line. The whole code block is actually one line this one as well. It is also displayed how it uses the best values from the *tuneOutSVM*, this is were gamma and cost is stored. They are in line 4 and line 5 reused in the

classifier, this is done by fetching the best parameters from *tuneOutSVM* for both gamma and cost. It is also stated which core to use, as in this project it is *libsvm*. This is also a parameter that has to be set in order for the classifier or model to be used afterwards. There is also a parameter saying cross=10, this is the cross validation talked about in the previous paragraph. To be able to use the classifier or model at a later time, the probability is set to be *TRUE*. When doing the actual prediction of the testing data set, the classifier is given as a parameter and for it to use the *svmClassifier* as the model is called the probability option has to be set to *TRUE*. There is also some more lines in the script, but the ones that are the most important for the algorithm is displayed in this section. The rest of this scripts and all of the other scripts used in the project can be found in the appendix. The scripts will have names indicating what it does and which of the scripts it is.

4.5.4 Script four

As with the k-NN algorithms, when making the multiclass classification script able to run the binary classification was not to hard to figure out how to write. All of the multiclass modifications is removed and the data set is changed to binary data set. When removing the lines in the multiclass classification the binary worked almost straight away. There are also some pointers that are set to point to column 42 instead of 43 as in the multiclass classification. There are also some small other modifications made to make the binary SVM script work as it should. The differences can be seen when comparing the scripts in the appendix. This binary script should also run faster than the multiclass classification due to it having fewer classification options. There is only anomaly and normal that is should classify the different events or entries from the data set as, just as in the one doing k-NN binary classification.

4.5.5 The timing of experiment one and two

The timing of the experiments is done using the system time in R. This is easily done by setting start time and end time using the command for system time in R, and then take end time minus the start time. This will give the time spent running the scripts. So in each of the scripts at the start and at the end it can be seen a command or variable being set to start and end time. These will time how long time it takes to run the whole script. The only concern with this way of timing is that it also do some modifications and so in these scripts. So the timing will be on running the whole script not just the algorithms. The process of modifying the data set do not take to long, and it is equal the amount of work done when it comes to modifications. How the system time command is used in the scripts will be listed in the code block called Measuring time usage.

Measuring time usage.

```
start.time <- Sys.time()
....
The script ...
....
end.time <- Sys.time()
time.taken <- end.time - start.time
time.taken
```

The timing of the script as can be seen from the code block stores the time when the script is initiated and when it exits the program. This is done by running the command *Sys.time* and it is stored to the two variables called *start.time* and *end.time*. These are used in the variable named *time.taken*, what *time.taken* does as can be seen from the code block, is to store the difference between *end.time* and *start.time*. After this is done it can be called upon using the variable name *time.taken*. This will then give as output how long the time difference is.

4.5.6 How the resource usage is measured

The resource usage will be monitored and measured using a program that gives resource usage. This program will be used by accessing the server via a SSH connection to the server. Then monitoring the process of the R session that is created for the user logged into RStudio is done by using pidstat. Pidstat can monitor a specific process by using its process id or pid. There is made a script that will monitor the scripts usage of resources. The script is run while the R session is active, this is as long as the user has a process or is logged into R.

Chapter 5

Results II: The experiments and analysis

5.1 Overview

In this chapter all of the results from the experiments will be presented. The results will be presented using different kinds of plots, bar charts, tables and some text to explain the different presentations. This chapter should give a overview of how the different experiments performed. The comparison of the experiments and so will mostly take place in the discussion chapter. Whilst in this chapter the analysis of the data gathered data from the experiments will be analyzed. The results from the different experiments will be gone through and explained thoroughly. The graphs, plots and other visual presentations will be walked through and key numbers and findings will be mentioned. The objective in this chapter is to present what the outcome of the tests or experiments done. It will give some indications that will be further explained and later in the discussion. It will be used to discuss which of the algorithms that had the best overall performance in the experiments.

There is a total of four experiments as explained earlier. The four experiments will be performed in the same order as planned and the results of each and every experiment will be analyzed. Sub experiments are done in experiment one and experiment two, these will go by the name of the algorithm and the way it is classified. So if the algorithm is SVM and the classification is binary it will be under the section experiment one and have the name SVM binary classification. This will be the case in the two first experiments that is going to measure the accuracy of the algorithms. The two first experiments will solely look at the accuracy of the experiments, and it will only present the numbers that can be linked to the accuracy of the algorithm. The two next experiments will look at a different aspect of machine learning or the algorithms, which is how efficient the algorithms are in time and how much resources they use when running. When saying resources, what is actually meant is the CPU usage. All of these results will be further discussed and a conclusion will sort of become clear in the

discussion. So there will not be any conclusions made in this chapter, other than analyzing the experiments.

All of the experiments have been performed and the results can be considered as pleasing. There is always some room for improvement, and there could probably have been some improvement when optimizing the different parameters and doing the experiments using the power of a GPU server and so on. Then some might say that using GPU servers would be to expensive for small or not so big businesses, and that might be the case for many corporations. So saying that these results of the experiments has been pleasing could be backed up by this. There has been good accuracy on both of the algorithms, and the parameters has been optimized by testing with ranges of values before it was set as the best values. When doing the experiments the different parameters is tested and the best will be used as the parameter that goes into the model. This model is then again a parameter in the prediction as explained in the first result chapter.

5.2 Experiment one

The first experiment done in the project is the binary classification. The scripts were made in a different order, but the actual experiments are done according to the plan. The binary classification is done to compare the binary and multiclass classifications, the comparison will take accuracy, time consumption and resource usage into consideration. Measuring how good the binary classification using the different algorithms should go faster than multiclass classification in theory. Saying that it should be faster in theory, is not a conclusion but an assumption. They might be equally fast in the project and it will be elaborated more in the fourth experiment.

Binary classification is when using just two different classes that it can classify the traffic as. In the project the two different classes the data set contains of is anomaly and normal, so it is either an attack or it is considered as normal traffic. When having fewer classes that the algorithm can classify the traffic into, it should be faster than the one having to classify traffic into five different classes. But this is yet to be seen after all of the experiments has been performed.

5.2.1 SVM Binary Classification

SVM is the first algorithm that was used in the experiments, it had high numbers when looking just at the accuracy of the prediction done based on the model of the algorithm. As can be seen from figure 5.1, the actual attacks and the prediction was close.

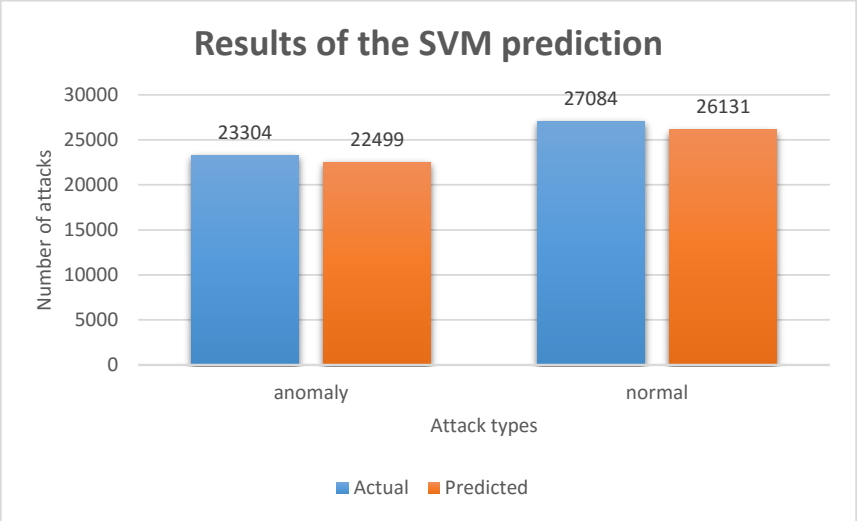


Figure 5.1: This bar chart displays the difference between how many predictions that has been correctly classified using SVM binary classification. It also shows how many it actually is in the data set that is marked as attacks, that it should have recognized.

In figure 5.1 the original number of types of the attack or normal traffic from the data set is represented by *Actual* and is the bars to the left in both of the groupings. In both cases of normal and anomaly the prediction had a high accuracy, with just under a thousand classifications that is incorrectly classified. When the data set has approximately 50.000, having a little under 2.000 classified incorrectly is not a bad result. When doing the calculation on how many percent that is detected it has an overall accuracy on 96,5 %. That means if the accuracy is on 96,5 %, then the error rate is 3,5 %. It is not a 100 %, but it is very close and that is a acceptable results. At least when comparing it to the result of other researchers, and what accuracy rate they have presented in their papers. The performance of the algorithm can be seen presented in table 5.1.

Table 5.1: The accuracy of the algorithm in percentages

Correctly classified	Incorrectly classified
96,51 %	3,49 %

The SVM algorithm is known for having good results, and that goes for the project as well. It can be seen from table 5.1 that the results are promising, but it might not be the best compared to what other researchers have presented. It is up there with the research found on the field of study, but there are some research done that has even better results using SVM.

Table 5.2: SVM binary rates

TP	FN	TN	FP
96,55 %	3,45 %	96,50 %	3,50 %

Table 5.2 displays a table with information that very often is used in papers using machine learning techniques to do anomaly detection or other classifications. It is based on the confusion matrix, a confusion matrix is the output of the prediction. It is actually just numbers presenting how the prediction went and the percentages have to be made manually. This is done by following a standard that explains how to calculate the different rates. *TP* is True Positive rate, and it tells how many of the anomaly that were correctly classified. *FN* or False Negative rate is how many of the anomalies that were classified as normal traffic. *TN* or True Negative rate is all of the predictions classified correctly as normal traffic. *FP* or False Positive rate is all of the normal traffic classified as anomaly. This gives a good picture on how good the accuracy of the algorithm is and could also give other information like possibility calculation can be calculated when having these numbers.

5.2.2 k-NN Binary Classification

K-NN or k-Nearest Neighbors is the next algorithm that were applied to the binary data set, in order to perform binary classification. This algorithm showed great results as well, with high percentages on accuracy. This can be seen from the bar chart in figure 5.2

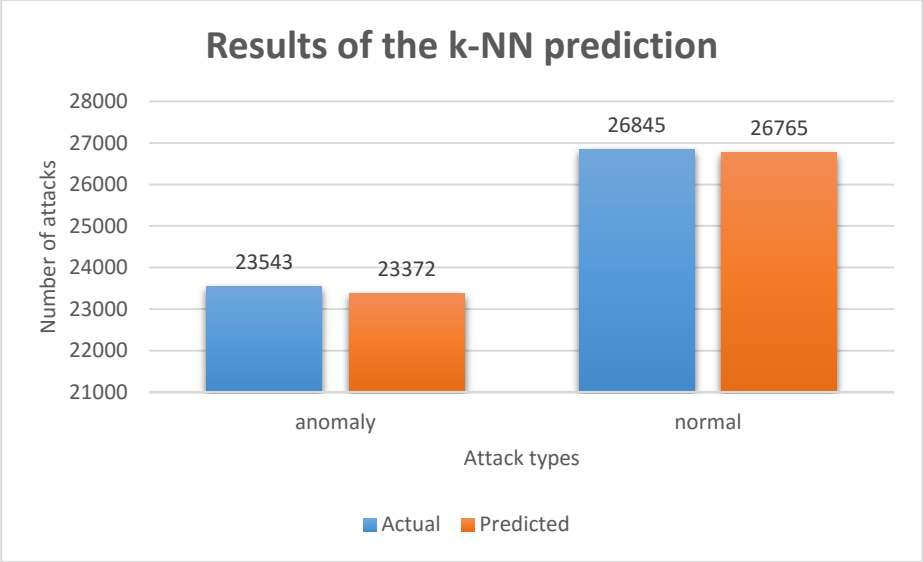


Figure 5.2: This bar chart displays the difference between how many predictions that has been correctly classified using k-NN binary classification. It also shows how many it actually is in the data set of different types of attacks, that it should have recognized.

This follows the same presentation principles as the figure 5.1 using SVM, it has the same labels and has anomaly and normal as the different classes. The bars displayed is also represented as the one to the left in the grouping is the actual number of attacks that are in the data set. The one to the right is the predictions based on the model made by the script using optimized parameters as input. When looking at the number from the bar chart it is easy to see that the accuracy is very high in this experiment. Table 5.3 displays the key numbers in percentages. This table shows that the accuracy when using the test set with binary attacks, it has an almost perfect score.

Table 5.3: The accuracy of the algorithm in percentages

Correctly classified	Incorrectly classified
99,49 %	0,51 %

With an accuracy of 99,49 % k-NN is very close to a perfect classification, it has less than a 100 incorrect classifications. This is a great result and when comparing it to what other researcher have gotten as results, it is in the top of the papers found and referred to in this paper.

Table 5.4: K-NN binary rates

TP	**FN**	**TN**	**FP**
99,27 %	0,73 %	99,70 %	0,30 %

Table 5.4 gives a very clear picture on how accurate the algorithms prediction based on the model have performed. When using k-NN the True positive and True negative values are very high, which indicates that the algorithm did a good job classifying the test data. The concept behind True positive, True negative and so on is explained in the previous experiment.

5.3 Experiment two

The second experiment is the one using multiclass classification, which will be more like a real life scenario. When saying real life scenario, what is meant is that there will be more then two classes the classifier can classify the data into. It will be a total of five different classes, which is mentioned earlier. These classes are: DoS, Probe, U2R(User to Root), R2L(Root to Local) and Normal. For the algorithm this means that it has to recognize the patterns and classify it as one out of the five different classes. This will be a more complicated process then just to classify it as either true or false, as with binary classification. Even when using 60 % of the data set for training, it is not given that the all of the different attacks will be in the

training data. This is due to the amount of samples of each attack. DoS, Probe and Normal are frequently repeated in the data set, but samples of the attacks R2L and U2R does not appear as often. Therefore it can be hard for the model to adapt to these attacks and learn them, while being tested.

In experiment two there are also some important modification as mentioned in the planning of the project and in the previous result chapter. To mention a few of the modification needed, there has to be added a column in the data set and it has to looped through in order to group all of the 22 different attacks it contains. There will also be necessary to changed some lines of code in order for the scripts to run as they should. This is all describe in the earlier chapters and are not the focus in this chapter. The most important changes is that it will use five classes, the data set has to be changed into the correct data set and then modified for this to work.

5.3.1 SVM Multiclass Classification

The first experiment done in experiment two is the one using the algorithm SVM to do multiclass classification. This will be done using supervised learning, a subcategory of machine learning as explained in the background chapter. This is what is used in both of the two first experiments. Meaning the experiments where the algorithms are applied, not the measuring of time and resource usage. What that supervised learning is shortly explained is that the algorithms used, when being learnt it knows what the attack is and it learn by grouping the different attacks and recognizes the patterns from the group of attacks. So in an essence it knows what to look for by learning the patterns of the classes beforehand and then looking for the patterns in the test data set.

All of the above is done in this experiment, it will be used machine learning techniques(Supervised learning) using the algorithm SVM. The algorithm SVM uses the hyperplanes as classifiers and support vectors as explained earlier in the background chapter. And in this experiment the algorithm did not have the best performance when looking at the accuracy numbers in figure 5.3 where the performance of SVM using mulitclass classification is presented. As can be seen from the bar chart it is a bit different from the first experiment, and this is due to the multiclass classification. It has five 10 bars in the chart, there 2 bars for each attack. The first of the two is the one indicating how many there are in total in the data set of that specific attack. The second bar is the one indicating how many that were actually classified as that specific attack. The bars at R2L and U2R is not missing there is just so few of these compared to the others. There should be three R2L, but it does not classify any as it and there should be 13 U2R but it only classifies 8 as U2R. And looking at the number that should be of R2L i.e. compared to the number of Normal traffic, it becomes clear that the bars will end up small on the ones with so low numbers.

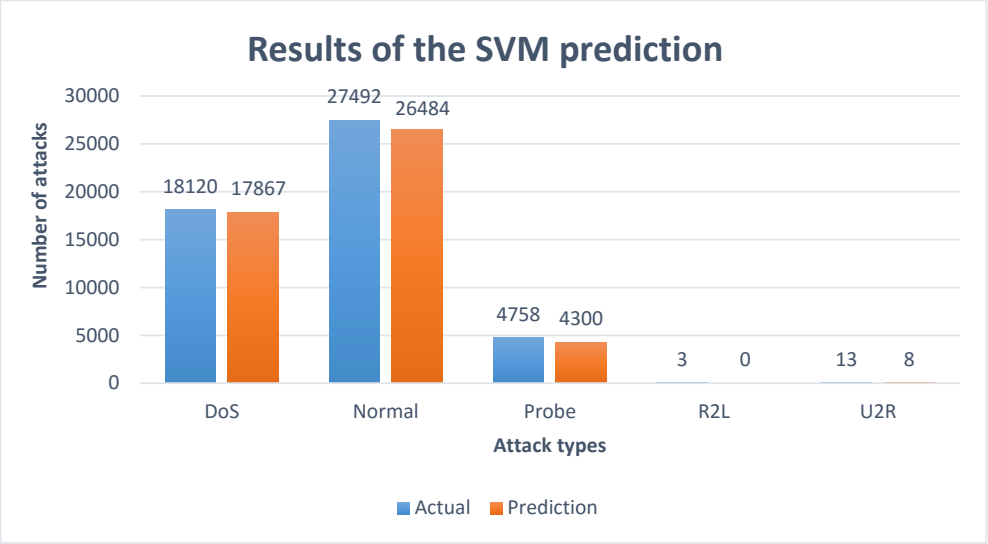


Figure 5.3: This bar chart displays the difference between how many predictions that has been correctly classified using SVM multiclass classification. It also shows how many it actually is in the data set of different types of attacks, that it should have recognized.

The overall classification is not the best, it has pretty good results on the ones like DoS, Normal and Probe. But in the two other cases, which can be critical to the overall performance of the algorithm, it does not live up to its expectations. On R2L it does not recognize any at all which brings down the overall performance a lot. The class U2R has at least an accuracy of 61,5 %, so it will not bring the overall accuracy that much. The overall performance can be seen in the table 5.5 bellow. As mentioned briefly in the

Table 5.5: The accuracy of the algorithm in percentages

Correctly classified	Incorrectly classified
69,37 %	30,63 %

beginning of this section describing the experiment, there are some other factors that can weigh in on the accuracy or performance of the algorithm. There are certain factors like the splitting of the data set. There are different amount of samples from the different attacks, and since they are grouped into five classes instead of 22, there could be some mistakes due to that. The data set with 22 different attacks contains certain attacks and these can have different values in the columns that is used. So even though there should be some in the training set, they might be classified as normal traffic due to the model not recognizing the specific attack since it might never have seen that specific pattern. This a bottleneck that is hard to come around when doing the classification like it is done in this project.

Table 5.6: The confusion matrix for SVM multiclass classification

	DoS	Normal	Probe	R2L	U2R
DoS	17867	134	118	1	0
Normal	359	26484	241	396	12
Probe	144	313	4300	1	0
R2L	0	0	3	0	0
U2R	0	5	0	0	8

What the table 5.6 displays is how many samples the algorithm classified correctly and incorrectly. The columns are aligned in a way that shows the correctly classified samples in the corresponding columns. So the column DoS which is the first both horizontal and vertical, way to the left in the table, shows the correctly predicted DoS samples. The column that is talked about is the one containing the number 17867. The others in the same row horizontally are the ones that should have been classified as DoS, but were not.

5.3.2 k-NN Multiclass Classification

There are many equal modifications that needs to be done in order to do multiclass classification using k-NN. K-NN works in a different way then SVM as explained earilier in this thesis. It uses the majority vote of the K-Nearest Neighbors to classify the outcome of the next classification. The algorithm is based on the Nearest Neighbor algorithm, but instead of just looking at just the nearest neighbor it looks at k of the nearest neighbours. K can be any given value, and there are many theories about what k actually should be. Some say that it should be somewhere around the square root of entries or samples in the data set, while others says that they used the optimized value of k. In this project k has been optimized using the *tune.knn()*, which is a function that returns the best values of k, when specifying a range of what k can be. As can be seen from figure 5.4, this gave some very good results.

The results that are presented in a bar chart follow the same order and idea as the one presenting the SVM multiclass classification. The actual number of the different attacks is the one bare to the left in the groupings, this is displayed in order to see how good the algorithm performed. The second bar in the grouping of attacks shows how many it managed to classify correctly. In this testing set, which the 40 % partitioning of the data set there are more of the R2L samples as can be seen from the diagram. In the SVM multiclass classification there were three samples of the attack in total, in this test data set there are 405 in total. This is due to the partitioning being done for each run of an algorithm, so the data sets both the training and the test data set will almost never look alike. There is a function that chooses the partitioning and the order of the data set, this is randomized and the order of the samples are also shuffled around due to the cross validation done to the data set. These are the reasons for the different

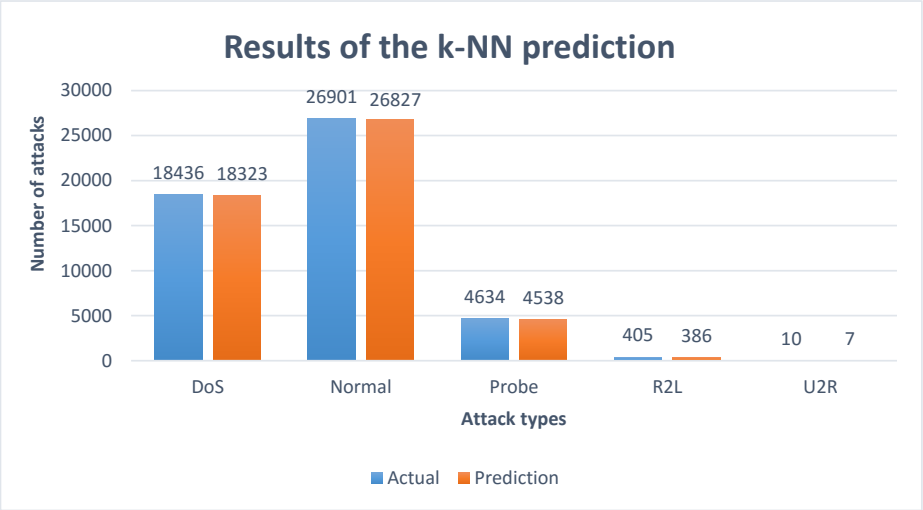


Figure 5.4: This bar chart displays the difference between how many predictions that has been correctly classified using k-NN multiclass classification, and how many it actually is in the data set of different types of attacks, that it should have recognized.

numbers in the diagram in the experiments. The test data is always 40 % of the original data set, whilst the training data set is always 60 %, but the order of the samples can be different each time since it randomly splits the data set into a training and testing data set.

Table 5.7: The accuracy of the algorithm in percentages

Correctly classified	Incorrectly classified
92,47 %	7,53 %

As table 5.7 displays the overall accuracy was at about 92,47 % and the incorrectly classified samples were then only at 7,53 %. When using multiclass data there cannot be made a table in the same matter as with binary data, which has the true positive, true negative and so forth. To make a table like that with the results gained from the multiclass predictions can be done, but then it would have been to compare all against one. Instead the confusion matrix will be displayed as in table 5.8

Table 5.8: The confusion matrix for k-NN multiclass classification

	DoS	Normal	Probe	R2L	U2R
DoS	18323	22	91	0	0
Normal	17	26827	33	12	12
Probe	30	65	4538	0	1
R2L	0	19	0	386	0
U2R	0	3	0	0	7

What the table 5.8 displays is how many samples the algorithm classified correctly and incorrectly. The principles behind the confusion matrix are explained in the SVM example.

5.4 Experiment three

In this experiment the time usage will be measured, this is a very important factor as well. The time being of importance is nothing new, everything new should run fast and if not have an exceptional result if does not run fast. What is done to measure how fast the different scripts and algorithms run is to fetch a time stamp from the R server. This is done when first in the script and then when everything is finished. The time stamp is fetched by using a command that queries the system for its time stamp, this command or line of code is as mentioned in the Result I: *Sys.time()*. This is then stored as *start.time* and *end.time*, before it is compared and the time usage is displayed by taking *end.time - start.time*. This is all explained even better in the previous chapter, but just as a reminder this is what done in order to measure the time usage.

The expected results were at least that the binary would take lesser time than the multiclass, due to it having fewer classes to classify the samples into. Unexpectedly this was not the case at all, both the binary and multiclass classification took the equal amount of time when looking at the same algorithm. The two algorithms SVM and k-NN, used a very different amount of time. One of them had a run time on the scripts at about three and a half hours, while the other used a little over one and a half hours. The fact that one of them used so much more time then the other is crucial. There is approximately 126.000 samples in the data set, as mentioned earlier. When looking at the amount and the time one could say that 126.000 samples processed in a little over one hour is acceptable, but three hours on processing the data set is a very long time. To compare this to the real world if the device used is connected to Internet and has possibilities to use SSH, 126.000 lines in a log can be quickly produced in a very short time. Big companies might have over 126.000 of lines in their logs after just an hour with traffic, so the time can be a issue at least when running the algorithm with the specification used in this project. There would have to be added possibilities for multi-core or GPU driven processing of the data.

The algorithm that used the least amount of time was k-NN, it used a little under one and a half hour as can be seen from table 5.9. The SVM classifier used over three hours to process the whole data set. This is the time it takes for the model to learn and test it, so it would be faster if only the test was performed and not the training as well.

Table 5.9: The time used by the different scripts

Algorithm and data set	**Time used by the script**
k-NN binary	1 hour and 24.6 minutes
k-NN multiclass	1 hour and 25.3 minutes
SVM binary	3 hours and 6 minutes.
SVM multiclass	3 hours and 32.1 minutes.

As can be seen from the table there is a great difference between the amount of time used by the algorithms, both of them are optimized by the script and other processing and modifications should be the same in all of the scripts. There is only the adding of the column in the multiclass scripts that is the most important difference. This process does not take a long time and does not have a big impact on the time used. This process uses about half a minute to finish running, so this is not something that will affect the end result.

5.5 Experiment four

Experiment four is the experiment that measures the resource usage. Since R studio or R server only assigns one core to each R session this is easy to do. When the user logs into R studio using the Web GUI(Graphical User Interface) it assigns the user a R session that can easily be monitored by running a simple scripts that fetches the output i.e. *pidstat* or *top*. When monitoring the CPU usage, (which is the resource that will have anything interesting to say about the resource usage when running the script) it becomes very clear that the resource usage is constant. This goes for both of the algorithms, the R session is assigned one core and runs the process using only one. This one core spikes immediately up to 100 % CPU usage and stays at that level until the script is finished. When looking at the CPU usage including all of the other cores it uses only 0,9 % of the CPU power available. For the R server to use multiple cores one would have to optimize the configuration, and it is not possible to use without this configuration. Just as with GPU computation as mentioned in previous chapters.

The only issue experienced with only having one core assigned to the process is that the web GUI freezes or hangs a bit when running the actual model, parameter optimisation and prediction. This is not a big issue, but the GUI becomes unresponsive and one have to be patient. When trying to

access the GUI it says that it take to long time to get a response, this is due to the script demanding all of the resources available. This can be frustrating but that is just how the R Server works when using a single core, and all of its power is used to compute the different results for the scripts.

The resource usages was not a great measurement to look at when trying to differentiate the two algorithms since they both spikes up to a 100 % and stays at this amount of resource usage throughout the running of all the scripts. So this experiment will not have an affect, when determining which of the algorithms that has the best overall performance.

Part III

Conclusion

Chapter 6

Discussion

The aim of this thesis has been to compare the overall performance of two different classification algorithms, when doing anomaly detection using machine learning techniques. This is what has been done and described in the previous chapters Approach, Results I, Results II and will in this chapter be discussed. There will also be discussed how the results of the experiments has affected the end result that will be discussed in this chapter. The experiments are designed in accordance with a purposed answer to the problem statement.

6.1 Problem statement

The problem statement is what the whole thesis is built on, and is the reason behind this project. The problem statement gives a clear problem or question, that is attempted giving an answer to in this thesis. The goal of this thesis is as stated in the problem statement, to try and come up with an answer to which classification algorithm who has the best overall performance. The problem statement as listed in the introduction of this thesis is:

Comparing the two algorithms SVM and k-NN, to determine which of the algorithms that yields the better overall performance, when doing anomaly detection using machine learning techniques

The parameters that determines which of the algorithms that has the best overall performance is:

- Classification performance
- Time consumption
- Resource consumption

These parameter were determined according to what the focus is in papers that have done similar studies, using classification algorithms. In these papers there are mainly focused on the detection rates, like i.e. true positives, false positives etc. There are almost never mentioned any thing

about the time consumption and resource consumption, in the papers found on the field of study. There are also some that also mentions the time consumptions but there are very few. The main focus in the papers found is as said on the classification rates/detection rates. If there should be any possibility for machine learning to be implemented into network security, then time is of importance. The same is the resource consumption in a lager scale, where there is the need for multiple cores for doing the classification and so on. These parameter was therefore found to interesting metrics that could determine the overall performance of the algorithms.

In this thesis there is made four different implementations of the algorithms. These implementations are suggestions to how they can be implemented, and get results that should be optimized. If the actual results can be considered representable for the algorithms still have to be determined. This will be further discussed in the algorithms and experiments part, along with some other considerations. There are several studies on both k-NN and SVM individually, but a comparison of just the two is not common. There are some papers that have compared a lot of algorithms, and this may not give the best results. When having to implement ten algorithm the focus on optimization might not be as important as just making them run as they should. In the papers where they study one of them the optimization of the algorithms is of great focus. That is why the optimization has been of focus in this thesis as well.

6.2 Algorithms and experiments

The outcome of the experiments gave a good indication on which of the two algorithms that performed best, when looking at the overall performance. The experiments were supposed to give an indication on how well the both of the algorithm performed and the results presented would give an indication. There results is open for discussion, and might not be conceived as valid by other researchers. The standards and thumb of rules found in other papers and the guides written on the topic.

6.2.1 Experiment one: Binary classification

The binary classification was done to see if this was a faster way to classify and to see how good the accuracy was compared to the multiclass. Implementing the algorithms using the binary classification did not demand to much time. This was due to the multiclass classification implementation of the algorithms were already in place. There were some small issues that had to fixed, but these were not to hard to fix when having understood how the algorithm packages worked. The binary classification gave some very good results on both of the algorithms. K-NN had the highest accuracy with $\approx$ 3 % higher accuracy then SVM. Both of the algorithms had a detection rate that were over 95 % which is very good results. The k-NN algorithm had a tremendously high accuracy with just a

few classification made incorrectly, and a overall accuracy at $\approx$ 99,5 %.

6.2.2 Experiment two: Multiclass classification

The second experiment also gave some good results, there was a bigger gap between the two algorithms when using multiclass data set. This even though narrowing it down to five classes instead of 22 classes which the original data set has. It could also be one of the reasons why one of them did not perform as good as expected, as explained in the Result II chapter. Again the k-NN algorithm was the one that performed the best with an accuracy at $\approx$ 92,5 %. The SVM had a disappointing accuracy at $\approx$ 69,5 %, which can not be concluded as a good result. There are other researchers that has had very good results using the algorithm, but many does not clarify whether or not the data set is binary or multiclass. There is also the fact about the data set having the same modification whether it is the binary or the multiclass, other than the class that is going to be predicted. This information is something that can indicate that the algorithm when used to do multiclass classification can struggle to get the desired results. Since when used to do binary classification it has an accuracy at about 96 %, while just changing the class it is going to predict into five classes instead of two, the accuracy drops to 69 %. This could be due to some error made by the author and may not be representable for the algorithm if an mistake has been made, but then again it is surprising that it preforms so well with the binary data. All of the parameters as said has been optimized for all of the algorithms using the *tune.svm* and *tune.knn* so this should not be the problem, but it might be.

6.2.3 Time consumption

In the approach there were made an assumption about the binary data set. This assumption was that the binary classification would be processed faster then the multiclass. This assumption was based on the facts that the binary data set contains two classes while the multiclass contains five classes. When having more classes to compare the patterns too, even though there are equal amount of samples would normally indicate that there would be used much more time. This were not a correct assumption, they were almost equally fast when running the binary and the multiclass classification. There were just some small differences, which are not too much time when looking at the total amount of time spent on the classifications. The SVM used half an hour more on the multiclass then the binary, so in this case the assumption was right. The k-NN algorithm on the other hand, used one minute more on the multiclass than the binary.

6.2.4 Resource consumption

The resource consumption could not give a clear indication on which of the algorithms that used more resources than the other. As said in the results, the resource usage was constant at 100 % on the core assigned to

the R session and it was a total of 0.9 % of the CPU capacity available. The other resources measured like memory and so on could not give a good indication either, that is why they are not even mentioned in the results.

6.3 The project

The project as a whole has been a very interesting experience. It started out as a very complex project and the learning curve was steep. After gaining the information needed to understand how the algorithms actually works and how they can be applied and used in R. Then the complex parts of the project were more understandable and did not any longer look impossible to manage in the authors point of view. Machine learning is a very big and wide field of study, which just keeps on growing. It is used for a lot of things these days and there are researchers developing new ideas on how it can be used for even more than it used for today. Just understanding what machine learning is can take time, since it used for so many purposes and there are so many ways it can be used. So trying to apply machine learning techniques, when doing anomaly detection was a challenge to overcome. When trying to get the information needed to make the actual implementations, there were so many different suggestions on how and what to do technically. The practical part was the most time consuming part to understand and implement, when trying to find examples of what others had done in the matter of programming there were close to none answers. There might be some where the researchers has used another data set then what is being used in this project, but each data set needs to be optimized in different ways. Some data set does only have continuous values and some only categorical values. This is just some of the modification that has to be in place in order for the algorithms to work at all.

6.3.1 The problems encountered

There were several problems encountered throughout the project, some were easy fixes others were not. The authors knowledge when it comes to machine learning was at the beginning of the project very limited, so there were many small problems along the way and some that were harder to overcome. The most difficult problems were the ones that were with the developing, these were tough to problem solve. R studio does not give the best error output, and can sometimes be very confusing. Trying to Google the errors can give many different suggestions to what the errors can be about, and might make it even more confusing. Therefore it was very good to have some help from supervisor and others, and there is also some forums that can be of help if needed. On these forums there are many talented researchers and people that can be of great help.

The most difficult problems encountered in this thesis was the ones that

was about how to manipulate or modify the data set. The manipulation was very time consuming and could maybe have been avoided if having a better prior knowledge to the technology used. R studio was nothing new, but the knowledge in R circled mostly around making some simple plots. So advancing to more complex programming was a tough step up.

6.3.2 The plan

The projects initial plan that was outlined in the approach has been followed and gave results in the end, which was the desired outcome of the plan. The different steps that was sketched up by using a UML diagram was followed, and the plan worked almost as it was thought out. There were some problems that were a bit time consuming, but the planned was followed step by step in order to make the deadline. The plan did not contain any set time limits, the reasoning for this was that estimating how much time to spend on each step would hard. As said the prior knowledge was limited and estimating how much time it would take to complete each step would be a difficult task. The main goal which is outlined in the plan was reached and this was the most important task, otherwise there would be nothing to compare the algorithms to. If only one of the algorithms had been implemented one could have compared the results of binary and multiclass data sets, but fortunately this was not the case in this project.

6.3.3 Constraints

There were some constraints and limitations discussed in the approach about privileges and other minor constraints that may have an effect on the project. The only limitation or constraints that actually was experienced, were the ones about the prior knowledge to the technology and machine learning. The restricted privileges on the server used, did not turn out to be a actual problem. The supervisor had the privileges needed and could modify the server if or when needed. The only time the supervisor had to be contacted was to get a monitor software installed. This software was the *pidstat* program, that can be installed by installing the package *sysstat*. This was done immediately by the supervisor, so that was not a problem at all.

Other constraints and limitations that were mentioned in the approach were time, and how the time might be critical. This was fortunately not the case either. The planned steps were all finished in time, but as said there were some minor step-backs that demanded more time then what was expected. When fixing these issues or problems in one script, meant that the other scripts would not have the same issues.

6.4 Future work

There are so many different algorithms that could be used to do classification and in this thesis two of the most common classification algorithms were chosen to be compared. What is actually meant by most common, is that they are widely used in research. But then again there are not many comparisons of the just the two. Many chose to go with a whole bunch of algorithms to see more results, but the positive thing about just using two is that these can be optimized and are the only focus.

If there had been more time and the authors knowledge on this technology were better from the beginning of the project, there probably would have been more algorithms incorporated. There was a lot to learn in a not too long period of time, so having two working algorithms is an accomplishment in it self. Had there been more time the plan would have been to try and implement a Neural networks algorithm. This algorithm can also be implemented using the programming language R. There is also packages available to implement this algorithm in R, that makes it convenient to use R. This algorithm uses a bit different way of doing the actual classifying. This would have to be studied more in order for it to be implemented in this project. The idea of using a neural network algorithm to compare with the outcome of two commonly used classification algorithms would have been nice. Unfortunately the time ran fast and this was just an idea, that if there had been any time left for it to be done, it would have been implemented.

As can be seen from the results there is a bit of a gap between the results found in the binary versus the multiclass classification, when using the algorithm SVM. If this is a common problem could have been a study that could enlighten the results gotten in this paper. The parameters should be optimized when running the scripts. Another thing that is a little off about the result is that it performs so well when using the the binary data, but when shifting to the multiclass it gains a very low accuracy. So the research would gain a lot from studying this even more than it has been in this project. There has been attempted to do some research in order to see if this an error made by the author or if this is a common thing when using SVM. The answers found were very ambiguous, there were some researchers that stated that it was very good using binary data. Others said that there are so many different versions of the original SVM and that the different versions could have different outcomes.

Chapter 7

Conclusion

The main goal of this thesis was to implement anomaly detection using machine learning techniques. The study has utilized a machine learning technique called supervised learning. It was implemented through usage of two algorithms, k-NN and SVM which are both classification algorithms. The results gained from implementing these two algorithms were then compared in order to see which of them is best suited to perform anomaly detection in a network environment.

The network environment was simulated by using a data set containing samples of network traffic. The data set contains different indicators for multiple attacks blended in with normal traffic. There are labels linked to each sample, which makes it possible for the algorithms to differentiate between patterns of attacks and normal traffic. The algorithms are implemented using both binary and multiclass data. This was done in order to observe how well the algorithms performed, when exposed to both binary and multiclass data.

The experiments conducted in this thesis was developed in order to give a proposed solution for the problem statement. The parameters listed in the problem statement were: classification performance, time and resource consumption. The experiments conducted in this thesis is based on these parameters. However, conclusions drawn from the results in this thesis, needs further research to determine the validity and trustworthiness of the research presented.

All of results indicated that the algorithm k-NN had a better classification performance in all of the experiments, while also consuming less time than SVM. The resource consumption were equal for the two algorithms, so this parameter was not a factor of attention in the comparison. The results gained indicated a very high classification accuracy when using the algorithm k-NN, with a accuracy on 99,27 % with binary and 92,47 % with multiclass data.

Bibliography

[1] *GA-internet-security-threat-report-volume*. URL: https://www4.symantec.com/mktginfo/whitepaper/istr/21347932_ga-internet-security-threat-report-volume-20-2015-social_v2.pdf.

[2] *2016 Emerging Cyber Threats Report*. URL: http://www.iisp.gatech.edu/2016-emerging-cyber-threats-report.

[3] *McAfee*. URL: http://www.mcafee.com/us/resources/reports/rp-threats-predictions-2016.pdf.

[4] Benjamin Morin et al. "A logic-based model to support alert correlation in intrusion detection." In: *Information Fusion* 10.4 (2009), pp. 285–299.

[5] Abror Abduvaliyev et al. "On the Vital Areas of Intrusion Detection Systems in Wireless Sensor Networks." In: *IEEE Communications Surveys & Tutorials IEEE Commun. Surv. Tutorials* 15.3 (2013), pp. 1223–1237.

[6] Ismail Butun, Salvatore D Morgera, and Ravi Sankar. "A survey of intrusion detection systems in wireless sensor networks." In: *Communications Surveys & Tutorials, IEEE* 16.1 (2014), pp. 266–282.

[7] Arthur L Samuel. "Some studies in machine learning using the game of checkers." In: *IBM Journal of research and development* 3.3 (1959), pp. 210–229.

[8] Andres Munoz. *Machine Learning and Optimization*. 2014.

[9] Eric Weiss. "Biographies: Eloge: Arthur Lee Samuel (1901-90)." In: *Annals of the History of Computing, IEEE* 14.3 (1992), pp. 55–69.

[10] T.M. Mitchell. *Machine Learning*. McGraw-Hill international editions - computer science series. McGraw-Hill Education, 1997. ISBN: 9780070428072. URL: https://books.google.no/books?id=xOGAngEACAAJ.

[11] Andrew Ng. *Machine Learning*. URL: http://openclassroom.stanford.edu/mainfolder/coursepage.php?course=machinelearning.

[12] Michael Negnevitsky. *Artificial intelligence: a guide to intelligent systems*. Pearson Education, 2005.

[13] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[14] *Machine Learning: What it is and why it matters*. URL: http://www.sas.com/en_us/insights/analytics/machine-learning.html.

[15] Tom Roughgarden. *Coursera*. URL: https://class.coursera.org/algo-004/lecture/preview.

[16] *Kernel-Machines.Org*. URL: http://www.kernel-machines.org/.

[17] Bernhard Scholkopf et al. "Comparing support vector machines with Gaussian kernels to radial basis function classifiers." In: *Signal Processing IEEE Transactions on* 45.11 (1997), 2758
bibrangedash 2765.

[18] Marti A. Hearst et al. "Support vector machines." In: *Intelligent Systems and their Applications, IEEE* 13.4 (1998), 18
bibrangedash 28.

[19] Leif E Peterson. "K-nearest neighbor." In: *Scholarpedia* 4.2 (2009), p. 1883.

[20] Padraig Cunningham and Sarah Jane Delany. "k-Nearest neighbour classifiers." In: *Multiple Classifier Systems* (2007), pp. 1–17.

[21] Yihua Liao and V Rao Vemuri. "Use of k-nearest neighbor classifier for intrusion detection." In: *Computers & Security* 21.5 (2002), pp. 439–448.

[22] *KDD Cup 1999 Data*. URL: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[23] Maoguo Gong et al. "An efficient negative selection algorithm with further training for anomaly detection." In: *Knowledge-Based Systems* 30 (2012), pp. 185–191.

[24] Terry Brugge UC Davis. *KDD Cup '99 dataset considered harmful*. URL: http://www.bruggerink.com/~zow/gradschool/kddcup99harmful.html.

[25] Matthew V Mahoney and Philip K Chan. "An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection." In: *Recent Advances in Intrusion Detection*. Springer. 2003, 220
bibrangedash 237.

[26] Mahbod Tavallaee et al. "A detailed analysis of the KDD CUP 99 data set." In: *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications 2009*. 2009.

[27] Samir Kant Sahu, Saumendra Sarangi, and Sanjaya Kumar Jena. "A detail analysis on intrusion detection datasets." In: *Advance Computing Conference (IACC), 2014 IEEE International*. IEEE. 2014, 1348
bibrangedash 1353.

[28] A Habibi .L University of New Brunswick. *ISCX NSL-KDD dataset | UNB*. URL: http://www.unb.ca/research/iscx/dataset/iscx-nsl-kdd-dataset.html.

[29] R Studio. *About*. URL: https://www.rstudio.com/about/.

[30] Snehal A Mulay, PR Devale, and GV Garje. "Intrusion detection system using support vector machine and decision tree." In: *International Journal of Computer Applications* 3.3 (2010), 40 bibrangedash 43.

[31] Xin Xu and Xuening Wang. "An adaptive network intrusion detection method based on PCA and support vector machines." In: *Advanced Data Mining and Applications*. Springer, 2005, 696 bibrangedash 703.

[32] Ming-Yang Su. "Using clustering to improve the KNN-based classifiers for online anomaly network traffic identification." In: *Journal of Network and Computer Applications* 34.2 (2011), 722 bibrangedash 730.

[33] Ming-Yang Su. "Real-time anomaly detection systems for Denial-of-Service attacks by weighted k-nearest-neighbor classifiers." In: *Expert Systems with Applications* 38.4 (2011), 3492 bibrangedash 3498.

[34] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.

Appendices

Appendix A

The scripts developed

In this chapter all of the script developed and used in the thesis will be listed. These are developed by the author. These are all available on Github: https://github.com/HenkeV/AD_NSL-KDD

A.1 k-NN binary

Binary classification using the algorithm k-NN.

```
start.time <- Sys.time()
# reads the files:
options(warn = -1)
#kdd_train=read.csv(file="KDDTrain+.txt", sep = ",")
kdd_train=read.csv(file="KDDTrain+.arff", sep = ",")
#kdd_train=kdd_train[,-43]
# reads the names of the columns
colnames <- read.table("names", skip = 1, sep = ":")
# Sets the names on the trainingset
names(kdd_train) <- colnames$V1
# requires/installs the packages
require(class)
trainIndex <- createDataPartition(kdd_train$attacks,p=.6, list=F)
kddTraining = kdd_train[trainIndex,]
kddTesting = kdd_train[-trainIndex,]
kddTest = kddTesting
kddTrainingTarget = as.factor(kddTraining$attacks)
kddTraining=kddTraining[, -c(2,3,4,42)]
kddTesting=kddTesting[, -c(2,3,4,42)]


zeroVarianceFeatures <- sapply(kddTraining, function(i){
  if((is.numeric(i) & !any(is.nan(i)) & sd(i) >0) | is.factor(i)
  | is.character(i)) TRUE
  else FALSE

})

sapply(kddTraining, function(x)all(is.na(x)))
naValuesTest <-  function (x) {
  w <- sapply(x, function(x)all(is.na(x)))
```

```
  if (any(w)) {
    stop(paste("All NA values are found in columns", paste(which(w),
    collapse=", ")))
  }
}


naValuesTest(kddTraining)

knn.cross <- tune.knn(x = kddTraining, y = kddTrainingTarget,
k =1:20,tunecontrol=tune.control(sampling = "cross"), cross=10)

pred<-(knn(kddTraining, kddTesting, kddTrainingTarget, k = 3))
table(pred,kddTest$attacks)
predicted <- data.frame(Predictions=(pred))
predicted$Actual=c(as.character(kddTest$attacks))

predicted$accuracy <- 0
predicted$Actual=as.factor(predicted$Actual)
for(i in 1:nrow(predicted))
  if(predicted[i,1]==predicted[i,2]){
    predicted[i,3]="1"
  }else if(predicted[i,1]!=predicted[i,2]){
    predicted[i,3]="0"
  }
plot(as.numeric(predicted$accuracy[0:300]))
lines(predicted$accuracy)

end.time <- Sys.time()
time.taken <- end.time - start.time
time.taken
```

A.2 k-NN multiclass

Multiclass classification using the algorithm k-NN.

```
start.time <- Sys.time()
# reads the files:
options(warn = -1)
kdd_train=read.csv(file="KDDTrain+.txt", sep = ",")
kdd_train=kdd_train[,-43]
# reads the names of the columns
colnames <- read.table("names", skip = 1, sep = ":")
# Sets the names on the trainingset
names(kdd_train) <- colnames$V1
# requires/installs the packages
require(class)
# predicts with KNN
#knn(train = kdd_train, test = kdd_train, cl = class, k = 355)
kdd_train$type_attack <- 0
#kdd_train$class <- as.character(kdd_train$class)
# loops through and writes the correct class based on
# the subclass which is attacks
for(i in 1:nrow(kdd_train))
```

```
  if((kdd_train[i,42]=="smurf")|(kdd_train[i,42]=="neptune")|
  (kdd_train[i,42]=="back")|(kdd_train[i,42]=="teardrop")|
  (kdd_train[i,42]=="pod")|(kdd_train[i,42]=="land")){
    kdd_train[i,43]="DoS"
  }else if(kdd_train[i,42]=='normal'){
    kdd_train[i,43]="Normal"
  }else if((kdd_train[i,42]=="buffer_overflow")|
  (kdd_train[i,42]=="loadmodule")|
  (kdd_train[i,42]=="perl")|
  (kdd_train[i,42]=="rootkit")){
    kdd_train[i,43]="U2R"
  }else if( (kdd_train[i,42]=="ftp_write")|
  (kdd_train[i,42]=="guess_passwd")|
  (kdd_train[i,42]=="multihop")|
  (kdd_train[i,42]=="phf")|
  (kdd_train[i,42]=="imap")|
  (kdd_train[i,42]=="spy")|
  (kdd_train[i,42]=="warezclient")|
  (kdd_train[i,42]=="warezmaster")){
    kdd_train[i,43]="R2L"
  }else if((kdd_train[i,42]=="ipsweep")|
  (kdd_train[i,42]=="nmap")|
  (kdd_train[i,42]=="portsweep")|
  (kdd_train[i,42]=="satan")){
    kdd_train[i,43]="Probe"
  }


trainIndex<-createDataPartition(kdd_train$type_attack,p=.6,list=F)
kddTraining = kdd_train[trainIndex,]
kddTesting = kdd_train[-trainIndex,]
kddTest = kddTesting
kddTrainingTarget = as.factor(kddTraining$type_attack)
kddTraining=kddTraining[, -c(2,3,4,42,43)]
kddTesting=kddTesting[, -c(2,3,4,42,43)]




zeroVarianceFeatures <- sapply(kddTraining, function(i){
  if((is.numeric(i) & !any(is.nan(i)) & sd(i) >0) | is.factor(i) |
  is.character(i)) TRUE
  else FALSE

})


sapply(kddTraining, function(x)all(is.na(x)))
naValuesTest <-  function (x) {
  w <- sapply(x, function(x)all(is.na(x)))
  if (any(w)) {
    stop(paste("All NA values are found in columns", paste(which(w),
    collapse=", ")))
  }
}
```

```
73
74
75  naValuesTest(kddTraining)
76
77  knn.cross <- tune.knn(x = kddTraining, y = kddTrainingTarget,
78  k = 1:20,tunecontrol=tune.control(sampling = "cross"), cross=10)
79
80  pred<-(knn(kddTraining, kddTesting, kddTrainingTarget,
81  k = knn.cross$best.parameter))
82  table(pred,kddTest$type_attack)
83  predicted <- data.frame(Predictions=(pred))
84  predicted$Actual=c(kddTest$type_attack)
85
86  predicted$accuracy <- 0
87  predicted$Actual=as.factor(predicted$Actual)
88  for(i in 1:nrow(predicted))
89    if(predicted[i,1]==predicted[i,2]){
90      predicted[i,3]="1"
91    }else if(predicted[i,1]!=predicted[i,2]){
92      predicted[i,3]="0"
93    }
94  plot(predicted$accuracy[0:300])
95  lines(predicted$accuracy)
96  end.time <- Sys.time()
97  time.taken <- end.time - start.time
98  time.taken
```

A.3 SVM Binary

Binary classification using the algorithm SVM.

```
1   start.time <- Sys.time()
2   library(caret)
3   library(dplyr)          # Used by caret
4   library(kernlab)        # support vector machine
5   library(pROC)                   # plot the ROC curves
6   library(e1071)
7   # reads the files:
8   options(warn = -1)
9   kdd_train=read.csv(file="KDDTrain+.arff", sep = ",")
10  kdd_train=kdd_train[,-43]
11  # reads the names of the columns
12  colnames <- read.table("names", skip = 1, sep = ":")
13  # Sets the names on the trainingset
14  names(kdd_train) <- colnames$V1
15  # requires/installs the packages
16  require(class)
17  kdd_train$service = as.character(kdd_train$service)
18  kdd_train$service[kdd_train$service == "auth"] = 1
19  kdd_train$service[kdd_train$service == "finger"] = 1
20  kdd_train$service[kdd_train$service == "bgp"] = 2
21  kdd_train$service[kdd_train$service == "courier"] = 2
22  kdd_train$service[kdd_train$service == "csnet_ns"] = 2
23  kdd_train$service[kdd_train$service == "ctf"] = 2
```

```
kdd_train$service[kdd_train$service == "daytime"] = 2
kdd_train$service[kdd_train$service == "discard"] = 2
kdd_train$service[kdd_train$service == "domain"] = 2
kdd_train$service[kdd_train$service == "echo"] = 2
kdd_train$service[kdd_train$service == "ecr_i"] = 2
kdd_train$service[kdd_train$service == "efs"] = 2
kdd_train$service[kdd_train$service == "exec"] = 2
kdd_train$service[kdd_train$service == "gopher"] = 2
kdd_train$service[kdd_train$service == "hostnames"] = 2
kdd_train$service[kdd_train$service == "http_443"] = 2
kdd_train$service[kdd_train$service == "imap4"] = 2
kdd_train$service[kdd_train$service == "iso_tsap"] = 2
kdd_train$service[kdd_train$service == "klogin"] = 2
kdd_train$service[kdd_train$service == "kshell"] = 2
kdd_train$service[kdd_train$service == "ldap"] = 2
kdd_train$service[kdd_train$service == "link"] = 2
kdd_train$service[kdd_train$service == "login"] = 2
kdd_train$service[kdd_train$service == "mtp"] = 2
kdd_train$service[kdd_train$service == "name"] = 2
kdd_train$service[kdd_train$service == "netbios_dgm"] = 2
kdd_train$service[kdd_train$service == "netbios_ns"] = 2
kdd_train$service[kdd_train$service == "netbios_ssn"] = 2
kdd_train$service[kdd_train$service == "netstat"] = 2
kdd_train$service[kdd_train$service == "nnsp"] = 2
kdd_train$service[kdd_train$service == "nntp"] = 2
kdd_train$service[kdd_train$service == "pop_2"] = 2
kdd_train$service[kdd_train$service == "printer"] = 2
kdd_train$service[kdd_train$service == "private"] = 2
kdd_train$service[kdd_train$service == "remote_job"] = 2
kdd_train$service[kdd_train$service == "rje"] = 2
kdd_train$service[kdd_train$service == "shell"] = 2
kdd_train$service[kdd_train$service == "sql_net"] = 2
kdd_train$service[kdd_train$service == "ssh"] = 2
kdd_train$service[kdd_train$service == "sunrpc"] = 2
kdd_train$service[kdd_train$service == "supdup"] = 2
kdd_train$service[kdd_train$service == "systat"] = 2
kdd_train$service[kdd_train$service == "uucp"] = 2
kdd_train$service[kdd_train$service == "uucp_path"] = 2
kdd_train$service[kdd_train$service == "vmnet"] = 2
kdd_train$service[kdd_train$service == "whois"] = 2
kdd_train$service[kdd_train$service == "Z39_50"] = 2
kdd_train$service[kdd_train$service == "domain_u"] = 3
kdd_train$service[kdd_train$service == "ftp_data"] = 3
kdd_train$service[kdd_train$service == "http"] = 3
kdd_train$service[kdd_train$service == "IRC"] = 3
kdd_train$service[kdd_train$service == "ntp_u"] = 3
kdd_train$service[kdd_train$service == "other"] = 3
kdd_train$service[kdd_train$service == "red_i"] = 3
kdd_train$service[kdd_train$service == "smtp"] = 3
kdd_train$service[kdd_train$service == "tftp_u"] = 3
kdd_train$service[kdd_train$service == "urh_i"] = 3
kdd_train$service[kdd_train$service == "urp_i"] = 3
kdd_train$service[kdd_train$service == "X11"] = 3
kdd_train$service[kdd_train$service == "eco_i"] = 4
```

```
kdd_train$service[kdd_train$service == "pm_dump"] = 4
kdd_train$service[kdd_train$service == "ftp"] = 5
kdd_train$service[kdd_train$service == "pop_3"] = 6
kdd_train$service[kdd_train$service == "tim_i"] = 6
kdd_train$service[kdd_train$service == "time"] = 6
kdd_train$service[kdd_train$service == "http_8001"] = 3
kdd_train$service[kdd_train$service == "http_2784"] = 3
kdd_train$service[kdd_train$service == "harvest"] = 3
kdd_train$service[kdd_train$service == "aol"] = 3
kdd_train$service[kdd_train$service == "telnet"] = 7
kdd_train$service = as.factor(kdd_train$service)

kdd_train$flag = as.character(kdd_train$flag)
kdd_train$flag[kdd_train$flag == "SH"] = 1
kdd_train$flag[kdd_train$flag == "SF"] = 2
kdd_train$flag[kdd_train$flag == "S3"] = 3
kdd_train$flag[kdd_train$flag == "S2"] = 4
kdd_train$flag[kdd_train$flag == "S1"] = 5
kdd_train$flag[kdd_train$flag == "S0"] = 6
kdd_train$flag[kdd_train$flag == "RSTR"] = 7
kdd_train$flag[kdd_train$flag == "RSTOS0"] = 8
kdd_train$flag[kdd_train$flag == "RSTO"] = 9
kdd_train$flag[kdd_train$flag == "REJ"] = 10
kdd_train$flag[kdd_train$flag == "OTH"] = 11
kdd_train$flag = as.factor(kdd_train$flag)

kdd_train$protocol_type = as.character(kdd_train$protocol_type)
kdd_train$protocol_type[kdd_train$protocol_type == "tcp"] = 1
kdd_train$protocol_type[kdd_train$protocol_type == "icmp"] = 2
kdd_train$protocol_type[kdd_train$protocol_type == "udp"] = 3
kdd_train$protocol_type = as.factor(kdd_train$protocol_type)


kdd_train=kdd_train[, -c(1,7,8,9,11,18,19,20,21,22,34,35,36,37,38,
39,40,41)]

trainIndex <- createDataPartition(kdd_train$attacks,p=.6,list= F)
kddTraining = kdd_train[trainIndex,]
kddTesting = kdd_train[-trainIndex,]
kddTrainingTarget = kddTraining$attacks

# Alternatively method for categorical values:
# protocol_typeTraining=model.matrix( ~ protocol_type - 1,
# data=kddTraining)
# kddTraining$protocol_type=protocol_typeTraining
# serviceTraining=model.matrix( ~ service - 1, data=kddTraining)
# kddTraining$service=serviceTraining
# flagTraining=model.matrix( ~ flag - 1, data=kddTraining)
# kddTraining$flag=flagTraining
#
# protocol_typeTesting=model.matrix( ~ protocol_type - 1,
# data=kddTesting)
# kddTesting$protocol_type=protocol_typeTesting
# serviceTesting=model.matrix( ~ service - 1, data=kddTesting)
```

```
# kddTesting$service=serviceTesting
# flagTesting=model.matrix( ~ flag - 1, data=kddTesting)
# kddTesting$flag=flagTesting

zeroVarianceFeatures <- sapply(kddTraining, function(i){
  if((is.numeric(i) & !any(is.nan(i)) & sd(i) >0) | is.factor(i) |
  is.character(i)) TRUE
  else FALSE

})

sapply(kddTraining, function(x)all(is.na(x)))
naValuesTest <-  function (x) {
  w <- sapply(x, function(x)all(is.na(x)))
  if (any(w)) {
    stop(paste("All NA values are found in columns", paste(which(w),
    collapse=", ")))
  }
}
naValuesTest(kddTraining)

tuneOutSVM<-tune.svm(as.factor(attacks)~.,data=kddTraining,
    gamma = 2^c(-8,-4,0,4), cost = 2^c(-8,-4,-2,0),
    tunecontrol = tune.control(cross = 3, sampling = "cross"))
plot(tuneOutSVM, transform.x = log2, transform.y = log2)

svmClassifier=svm(as.factor(kddTraining$attacks)~ .,
    data=kddTraining,
    core="libsvm",kernel="linear",cross=10,
    gamma = tuneOutSVM$best.parameters$gamma,
    cost = tuneOutSVM$best.parameters$cost, probability=TRUE)

pred <- predict(svmClassifier, kddTesting)
predicted <- data.frame(Predictions=(pred))
predicted$Actual=c(as.character(kddTesting$attacks))

predicted$accuracy <- 0
predicted$Actual=as.factor(predicted$Actual)
for(i in 1:nrow(predicted))
  if(predicted[i,1]==predicted[i,2]){
    predicted[i,3]="1"
  }else if(predicted[i,1]!=predicted[i,2]){
    predicted[i,3]="0"
  }
plot(as.numeric(predicted$accuracy[0:300]))
lines(predicted$accuracy)

end.time <- Sys.time()
time.taken <- end.time - start.time
time.taken
```

A.4 SVM Multiclass

Multiclass classification using the algorithm SVM.

```
start.time <- Sys.time()
library(caret)
library(dplyr)         # Used by caret
library(kernlab)       # support vector machine
library(pROC)               # plot the ROC curves
library(e1071)
# reads the files:
options(warn = -1)
kdd_train=read.csv(file="KDDTrain+.txt", sep = ",")
kdd_train=kdd_train[,-43]
# reads the names of the columns
colnames <- read.table("names", skip = 1, sep = ":")
# Sets the names on the trainingset
names(kdd_train) <- colnames$V1
# requires/installs the packages
require(class)
# predicts with KNN
#knn(train = kdd_train, test = kdd_train, cl = class, k = 355)
kdd_train$type_attack <- 0
#kdd_train$class <- as.character(kdd_train$class)
# loops through and writes the correct class based on
# the subclass which is attacks
for(i in 1:nrow(kdd_train))
  if((kdd_train[i,42]=="smurf")|
  (kdd_train[i,42]=="neptune")|
  (kdd_train[i,42]=="back")|
  (kdd_train[i,42]=="teardrop")|
  (kdd_train[i,42]=="pod")|
  (kdd_train[i,42]=="land")){
    kdd_train[i,43]="DoS"
  }else if(kdd_train[i,42]=='normal'){
    kdd_train[i,43]="Normal"
  }else if((kdd_train[i,42]=="buffer_overflow")|
  (kdd_train[i,42]=="loadmodule")|
  (kdd_train[i,42]=="perl")|
  (kdd_train[i,42]=="rootkit")){
    kdd_train[i,43]="U2R"
  }else if( (kdd_train[i,42]=="ftp_write")|
  (kdd_train[i,42]=="guess_passwd")|
  (kdd_train[i,42]=="multihop")|
  (kdd_train[i,42]=="phf")|
  (kdd_train[i,42]=="imap")|
  (kdd_train[i,42]=="spy")|
  (kdd_train[i,42]=="warezclient")|
  (kdd_train[i,42]=="warezmaster")){
    kdd_train[i,43]="R2L"
  }else if((kdd_train[i,42]=="ipsweep")|
  (kdd_train[i,42]=="nmap")|
  (kdd_train[i,42]=="portsweep")|
  (kdd_train[i,42]=="satan")){
    kdd_train[i,43]="Probe"
  }
```

```
53
54  kdd_train$service = as.character(kdd_train$service)
55  kdd_train$service[kdd_train$service == "auth"] = 1
56  kdd_train$service[kdd_train$service == "finger"] = 1
57  kdd_train$service[kdd_train$service == "bgp"] = 2
58  kdd_train$service[kdd_train$service == "courier"] = 2
59  kdd_train$service[kdd_train$service == "csnet_ns"] = 2
60  kdd_train$service[kdd_train$service == "ctf"] = 2
61  kdd_train$service[kdd_train$service == "daytime"] = 2
62  kdd_train$service[kdd_train$service == "discard"] = 2
63  kdd_train$service[kdd_train$service == "domain"] = 2
64  kdd_train$service[kdd_train$service == "echo"] = 2
65  kdd_train$service[kdd_train$service == "ecr_i"] = 2
66  kdd_train$service[kdd_train$service == "efs"] = 2
67  kdd_train$service[kdd_train$service == "exec"] = 2
68  kdd_train$service[kdd_train$service == "gopher"] = 2
69  kdd_train$service[kdd_train$service == "hostnames"] = 2
70  kdd_train$service[kdd_train$service == "http_443"] = 2
71  kdd_train$service[kdd_train$service == "imap4"] = 2
72  kdd_train$service[kdd_train$service == "iso_tsap"] = 2
73  kdd_train$service[kdd_train$service == "klogin"] = 2
74  kdd_train$service[kdd_train$service == "kshell"] = 2
75  kdd_train$service[kdd_train$service == "ldap"] = 2
76  kdd_train$service[kdd_train$service == "link"] = 2
77  kdd_train$service[kdd_train$service == "login"] = 2
78  kdd_train$service[kdd_train$service == "mtp"] = 2
79  kdd_train$service[kdd_train$service == "name"] = 2
80  kdd_train$service[kdd_train$service == "netbios_dgm"] = 2
81  kdd_train$service[kdd_train$service == "netbios_ns"] = 2
82  kdd_train$service[kdd_train$service == "netbios_ssn"] = 2
83  kdd_train$service[kdd_train$service == "netstat"] = 2
84  kdd_train$service[kdd_train$service == "nnsp"] = 2
85  kdd_train$service[kdd_train$service == "nntp"] = 2
86  kdd_train$service[kdd_train$service == "pop_2"] = 2
87  kdd_train$service[kdd_train$service == "printer"] = 2
88  kdd_train$service[kdd_train$service == "private"] = 2
89  kdd_train$service[kdd_train$service == "remote_job"] = 2
90  kdd_train$service[kdd_train$service == "rje"] = 2
91  kdd_train$service[kdd_train$service == "shell"] = 2
92  kdd_train$service[kdd_train$service == "sql_net"] = 2
93  kdd_train$service[kdd_train$service == "ssh"] = 2
94  kdd_train$service[kdd_train$service == "sunrpc"] = 2
95  kdd_train$service[kdd_train$service == "supdup"] = 2
96  kdd_train$service[kdd_train$service == "systat"] = 2
97  kdd_train$service[kdd_train$service == "uucp"] = 2
98  kdd_train$service[kdd_train$service == "uucp_path"] = 2
99  kdd_train$service[kdd_train$service == "vmnet"] = 2
100 kdd_train$service[kdd_train$service == "whois"] = 2
101 kdd_train$service[kdd_train$service == "Z39_50"] = 2
102 kdd_train$service[kdd_train$service == "domain_u"] = 3
103 kdd_train$service[kdd_train$service == "ftp_data"] = 3
104 kdd_train$service[kdd_train$service == "http"] = 3
105 kdd_train$service[kdd_train$service == "IRC"] = 3
106 kdd_train$service[kdd_train$service == "ntp_u"] = 3
```

```
kdd_train$service[kdd_train$service == "other"] = 3
kdd_train$service[kdd_train$service == "red_i"] = 3
kdd_train$service[kdd_train$service == "smtp"] = 3
kdd_train$service[kdd_train$service == "tftp_u"] = 3
kdd_train$service[kdd_train$service == "urh_i"] = 3
kdd_train$service[kdd_train$service == "urp_i"] = 3
kdd_train$service[kdd_train$service == "X11"] = 3
kdd_train$service[kdd_train$service == "eco_i"] = 4
kdd_train$service[kdd_train$service == "pm_dump"] = 4
kdd_train$service[kdd_train$service == "ftp"] = 5
kdd_train$service[kdd_train$service == "pop_3"] = 6
kdd_train$service[kdd_train$service == "tim_i"] = 6
kdd_train$service[kdd_train$service == "time"] = 6
kdd_train$service[kdd_train$service == "http_8001"] = 3
kdd_train$service[kdd_train$service == "http_2784"] = 3
kdd_train$service[kdd_train$service == "harvest"] = 3
kdd_train$service[kdd_train$service == "aol"] = 3
kdd_train$service[kdd_train$service == "telnet"] = 7
kdd_train$service = as.factor(kdd_train$service)

kdd_train$flag = as.character(kdd_train$flag)
kdd_train$flag[kdd_train$flag == "SH"] = 1
kdd_train$flag[kdd_train$flag == "SF"] = 2
kdd_train$flag[kdd_train$flag == "S3"] = 3
kdd_train$flag[kdd_train$flag == "S2"] = 4
kdd_train$flag[kdd_train$flag == "S1"] = 5
kdd_train$flag[kdd_train$flag == "S0"] = 6
kdd_train$flag[kdd_train$flag == "RSTR"] = 7
kdd_train$flag[kdd_train$flag == "RSTOS0"] = 8
kdd_train$flag[kdd_train$flag == "RSTO"] = 9
kdd_train$flag[kdd_train$flag == "REJ"] = 10
kdd_train$flag[kdd_train$flag == "OTH"] = 11
kdd_train$flag = as.factor(kdd_train$flag)

kdd_train$protocol_type = as.character(kdd_train$protocol_type)
kdd_train$protocol_type[kdd_train$protocol_type == "tcp"] = 1
kdd_train$protocol_type[kdd_train$protocol_type == "icmp"] = 2
kdd_train$protocol_type[kdd_train$protocol_type == "udp"] = 3
kdd_train$protocol_type = as.factor(kdd_train$protocol_type)



# Remove NAs and zero values
kdd_train=kdd_train[, -c(1,7,8,9,11,18,19,20,21,22,34,35,36,37,38,
39,40,41,42)]
#kdd_train=kdd_train[, -c(2,3,4,42,43)]

trainIndex<-createDataPartition(kdd_train$type_attack,p=.6,list=F)
kddTraining = kdd_train[trainIndex,]
kddTesting = kdd_train[-trainIndex,]
kddTrainingTarget = as.factor(kddTraining$type_attack)

# Alternatively:
# protocol_typeTraining=model.matrix( ~ protocol_type - 1,
```

```
# data=kddTraining)
# kddTraining$protocol_type=protocol_typeTraining
# serviceTraining=model.matrix( ~ service - 1, data=kddTraining)
# kddTraining$service=serviceTraining
# flagTraining=model.matrix( ~ flag - 1, data=kddTraining)
# kddTraining$flag=flagTraining
#
# protocol_typeTesting=model.matrix( ~ protocol_type - 1,
# data=kddTesting)
# kddTesting$protocol_type=protocol_typeTesting
# serviceTesting=model.matrix( ~ service - 1, data=kddTesting)
# kddTesting$service=serviceTesting
# flagTesting=model.matrix( ~ flag - 1, data=kddTesting)
# kddTesting$flag=flagTesting

zeroVarianceFeatures <- sapply(kddTraining, function(i){
  if((is.numeric(i) & !any(is.nan(i)) & sd(i) >0) | is.factor(i) |
  is.character(i)) TRUE
  else FALSE

})
sapply(kddTraining, function(x)all(is.na(x)))
naValuesTest <-  function (x) {
  w <- sapply(x, function(x)all(is.na(x)))
  if (any(w)) {
    stop(paste("All NA values are found in columns", paste(which(w),
    collapse=", ")))
  }
}
naValuesTest(kddTraining)
tuneOutSVM <- tune.svm(as.factor(type_attack)~., data=kddTraining,
    gamma = 2^c(-8,-4,0,4), cost = 2^c(-8,-4,-2,0),
    tunecontrol = tune.control(cross = 3, sampling = "cross"))
plot(tuneOutSVM, transform.x = log2, transform.y = log2)
svmClassifier=svm(as.factor(kddTraining$type_attack)~ .,
    data=kddTraining,
    core="libsvm",kernel="linear",cross=10,
    gamma = tuneOutSVM$best.parameters$gamma,
    cost = tuneOutSVM$best.parameters$cost, probability=TRUE)

pred <- predict(svmClassifier, kddTesting)
table(pred,kddTesting$type_attack)
predicted <- data.frame(Predictions=(pred))
predicted$Actual=c(kddTesting$type_attack)

predicted$accuracy <- 0
predicted$Actual=as.factor(predicted$Actual)
for(i in 1:nrow(predicted))
  if(predicted[i,1]==predicted[i,2]){
    predicted[i,3]="1"
  }else if(predicted[i,1]!=predicted[i,2]){
    predicted[i,3]="0"
  }
plot(predicted$accuracy[0:300])
```

```
lines(predicted$accuracy)

end.time <- Sys.time()
time.taken <- end.time - start.time
time.taken
```