Intrusion detection with the K-nearest neighbour algorithm

A study on how well the K-nearest-neighbour algorithm performs in detecting network intrusions

William Sæbø
Master's Thesis Spring 2017
Intrusion detection with the K-nearest neighbour algorithm

William Sæbø

22nd February 2017
Abstract

With the proliferation of cyber-attacks, there is an increased interest among practitioners and in academy in using Machine Learning as a defense, detection and prevention tool. In this thesis, we study the efficiency of one of the most simple and yet efficient legacy Machine Learning algorithms, namely the K-nearest-neighbour algorithm, in detecting intrusions. We investigate how different portions of training data and the value of $k$ (i.e. the number of neighbors) might affect the classification performance in three different datasets. As a benchmark dataset, we will use the KDD cup dataset, but the algorithm will also be tested on the built in IRIS dataset. The findings in the thesis demonstrate that the KNN algorithm is quite accurate in predicting attacks despite its simplicity.
Contents

1 Introduction .................................................. 1
  1.1 Problem statement ...................................... 1
  1.1.1 Scope .................................................. 2

2 Background .................................................. 3
  2.1 Review of Intrusion Detection Systems ............... 3
    2.1.1 Types of IDS ...................................... 3
    2.1.2 IDS in research .................................... 4
  2.2 Review of Machine Learning ......................... 9
    2.2.1 Usage of Machine Learning ....................... 9
    2.2.2 Supervised techniques ........................... 10
    2.2.3 Unsupervised techniques ......................... 13
    2.2.4 Machine Learning usage in IDS ................ 13
  2.3 Confusion matrix ....................................... 14

3 Approach .................................................. 17
  3.1 Types of Machine Learning algorithms .............. 17
  3.2 Traditional IDS/IPS implementation ................. 18
    3.2.1 SNORT ruleset .................................... 19
  3.3 Technical part ......................................... 19
    3.3.1 KDD cup dataset .................................. 20
    3.3.2 Performance of algorithms ...................... 21
    3.3.3 R studio ........................................... 22
    3.3.4 Algorithms to be tested ......................... 24

4 Experiments ............................................... 25
  4.1 Experiment 1 ............................................. 25
    4.1.1 Steps .............................................. 25
    4.1.2 Preparing R ...................................... 25
    4.1.3 Short analysis .................................... 28
  4.2 Experiment 2 ............................................. 28
    4.2.1 Short analysis .................................... 28
  4.3 Experiment 3 ............................................. 29
    4.3.1 Short analysis .................................... 29
  4.4 Experiment 4 ............................................. 29
    4.4.1 Short analysis .................................... 30
  4.5 Experiment 5 ............................................. 30
4.5.1 Short analysis ........................................... 31
4.6 Experiment 6 ............................................. 31
  4.6.1 Short analysis .......................................... 31
4.7 KNN with KDD cup ......................................... 31
  4.7.1 Preparing KDD cup ..................................... 32
4.8 Experiment 7 ............................................. 32
  4.8.1 KDD with a smaller set ................................ 33
  4.8.2 Short analysis .......................................... 33
4.9 Experiment 8 ............................................. 34
  4.9.1 Short analysis .......................................... 34
4.10 Experiment 9 ............................................ 34
  4.10.1 Short analysis ......................................... 35
4.11 General observations ...................................... 35
4.12 Experiment 10 ........................................... 35
  4.12.1 Short analysis ......................................... 36
4.13 Experiment 11 ........................................... 36
  4.13.1 Short analysis ......................................... 36
4.14 Experiment 12 ........................................... 38
  4.14.1 Short analysis ......................................... 39
4.15 Experiment 13 ........................................... 40
  4.15.1 Short analysis ......................................... 41
4.16 Experiment 14 ........................................... 41
  4.16.1 Short analysis ......................................... 43

5 Discussion .............................................. 45
  5.1 KNN algorithm ............................................ 45
  5.2 Discussions of the experiments .............................. 46
    5.2.1 Experiment 1 .......................................... 46
    5.2.2 Experiment 10 ......................................... 46
    5.2.3 Experiment 2 .......................................... 47
    5.2.4 Experiment 11 ......................................... 47
    5.2.5 Experiment 7, 8 and 9 ................................ 49
    5.2.6 Experiment 3 .......................................... 50
    5.2.7 Experiment 12 .......................................... 50
    5.2.8 Experiment 4, 5 and 6 ................................ 51
    5.2.9 Experiment 14 .......................................... 52
    5.2.10 General observations .................................. 52
    5.2.11 Other algorithms ...................................... 52
    5.2.12 Training data ......................................... 53

6 Conclusion and future work .................................. 55
  6.1 Future work ............................................... 55
List of Tables

2.1 Example confusion matrix .............................................. 15

4.1 Experiment 10 prediction ............................................. 35
4.2 Experiment 10 facts .................................................... 36
4.3 Experiment 11 prediction ............................................. 36
4.4 Experiment 11 facts .................................................... 36
4.5 Experiment 12 prediction ............................................. 38
4.6 Experiment 12 facts .................................................... 39
Preface

I would like to thank all my friends and family for their outstanding support when writing this thesis. Writing a thesis is not an easy task, so without their support this task would have been much more difficult. Also a big thanks to staff at UiO and HiOA and particularly Prof. Anis Yazidi and Prof. Hårek Haugerud for helping me out with many questions during this process.
Chapter 1

Introduction

Intrusion Detection Systems (IDS) monitor network traffic and any suspicious or abnormal activity on a network. If suspicious or abnormal activity is detected then the network administrator is alerted. The IDS is usually situated on the outside of a firewall to prevent malicious network traffic getting from the Internet into an organization’s servers and computers. Even though IDS is essential in today’s network environment, they do have issues which require further research specially when it comes to zero-day attacks. In fact, legacy IDS are efficient in detecting known attacks for which a signature is available. However, they fail to detect new attacks, or, what is called zero-day attacks. A possible promising technique is to deploy Machine Learning. Machine Learning can accomplish two main tasks when it comes to IDS:

- efficiently distinguish between normal traffic and abnormal traffic based on historical data.
- is susceptible of learning different classes of attacks.

The field of Machine Learning has gained increased popularity and interest in the last decade. We see this for instance in spam filters where the email provider or client filters out spam messages. Machine Learning is also widely used in search engines and image classification and recognition. IDS is also a potential arena for Machine Learning. However, Machine Learning is not widely used in IDS commercially, despite the the fact that area is heavily researched.

1.1 Problem statement

The main aim of this thesis is to investigate the following question: "How does the value of K and different training data affect the KNN algorithm in different training sets?"

The thesis will also take a closer look at different concepts in intrusion detection and some other Machine Learning algorithms.
1.1.1 Scope

The scope of this thesis is to apply Machine Learning algorithms on the KDD Cup dataset. To do this the programming language R will be used. The advantages of using this dataset are that the dataset is already sorted and labelled and therefore one does not need to spend much time generating a new dataset. In addition to the already widely known Machine Learning algorithms, this thesis will also research some algorithms not widely known in the area of Machine Learning. That may provide some new information about other algorithms which may be appropriate for Machine Learning in intrusion detection. This thesis will also take a closer look at SNORT, which is a widely used IDS, and also use another built in dataset to compare the performance relative to the KDD dataset.
Chapter 2

Background

In this chapter, we review the two main concepts in this thesis, namely IDS and Machine Learning. Furthermore, we provide an overview over the legacy work on Machine Learning based IDS.

2.1 Review of Intrusion Detection Systems

An intrusion detection system (IDS) "is a device or software application that monitors network or system activities for malicious activities or policy violations and produces electronic reports to a management station" [17]. There are many types of IDS systems and one way to categorize them is as follows:

- System admin response: Active, passive
- Technology based: host-based, network-based, network behavior analysis, wireless based and mixed
- Detection methodology: Signature-based, behavior-based, stateful protocol analysis.

2.1.1 Types of IDS

The two main families of IDS systems are active and passive [14]. The main difference between them is that an active system is configured to intervene when suspicious activity is detected, whereas a passive system only gives us alert messages and it is more up to the sysadmin to take action on the messages. The advantage of an active IDS is, of course, that a sysadmin can "relax" in the sense that he or she can, in most cases, rely on the IDS taking action on suspicious activity in the network. Ironically, the biggest advantage of an active IDS may also be its biggest disadvantage. If a sysadmin does not check the IDS regularly (e.g. for updates) we can end up with an IDS which does not work as intended. Attackers always look for new ways to bypass the IDS and one can imagine the possibly disastrous consequences
if your **IDS** does not work.

The main advantage of a passive **IDS** is that a **sysadmin** has to stay on the alert all the time and has to take the proper action according to the alerts. You will always be sure that something will be done. However, a possible disadvantage is that a **sysadmin** may not have enough knowledge to take the proper action. Even worse, if a wrong action is taken, the consequences could be very serious.

**IDS** can also be subdivided into host-based **IDS** and network-based **IDS**. A host-based **IDS** runs on each individual host (for instance our own laptop with a firewall software). Network-based **IDS**, on the other hand, runs in the boundary of the network (think of a server running ).

Most companies run a combination of both (since virtually all versions of Windows nowadays come with a security kit included). Network-based **IDS** clearly has an advantage over host-based **IDS**. With a host-based **IDS** we cannot in an easy way discover attacks on multiple nodes. Network-based **IDS**, however, monitors traffic on the whole network and can easily discover attacks bound for multiple nodes.

One can also divide **IDS** into the methods they use to discover attacks. There are both signature-based **IDS** and behaviour-based **IDS**. Signature-based **IDS** discovers attacks using data from a huge database of known attacks. However, a disadvantage is that these databases must of course be maintained by someone and they might be behind when there are new types of attacks which are not yet added to the database. **SNORT** is probably the most well known signature-based **IDS** available. Behaviour based **IDS** on the other hand have patterns they recognize in the network to distinguish normal traffic from suspicious traffic.

As an example of a behaviour-based **IDS**, let us say you one day deal with a huge amount of data to upgrade some sort of software on many clients. This may trigger a false alarm on the **IDS**. The **IDS** learnt behaviour may be 100 megabytes of data pr day - but on the next day it is 5 gigabytes. This is of course a huge deviation and may trigger an alarm. In a smaller company the **sysadmin** more probably knows what is going on, but if the company is larger with many people working in the IT department, one **sysadmin** may not know. That person may then take some sort of action. Even if this scenario is somewhat unrealistic, it still illustrates a situation which may occur with a behaviour-based **IDS**

### 2.1.2 **IDS** in research

A brief introduction to **IDS** was covered in the section above. However, this topic is quickly evolving and there is a good amount of research on the topic. A good review paper on the topic is Liao [8] which covers a review on the topic in general. Already in the introduction the paper makes a clear distinction between **intrusion detection systems** and **intrusion prevention systems**. The distinction is that **IPS** has all the features the **IDS** has, plus the
capability to stop attacks. The paper also focuses on issues in virtual machines which is particular interesting since more and more data is stored in the cloud. The paper states three major detection categories which go into a bit more detail than the categories listed earlier. These are signature based detection, anomaly based detection, and stateful protocol analysis. Signature based detection relies on signatures to detect attacks. This can include a string or a pattern which correspond to an attack. Synonyms to signature based detection are knowledge based detection and misuse detection. The pros listed in the paper when it comes to signature based detection are that it is the simplest and effective method to detect known attacks and that it can analyse in detail based on context. One can translate this into a real world context. One can illustrate such context attacks with a real work example. An order for a pizza to be delivered at home is for instance a known "package" because it has the signature "pizza". However, a "package" which says that a thief is entering my home is one which we do not want since the signature is "thief". The cons with signature based detections according to the paper are that it is difficult to discover unknown attacks, that the technology has little understanding of states and protocols, that it is hard to keep signatures up to date and time consuming to maintain the knowledge. If an attack does not match one of the signatures in the database it is of course difficult to discover.

Anomaly based detection or behaviour based detection is covered briefly above, but a quick repetition is that it discovers deviations from a known behaviour. A deviation listed in the paper is failed login attempts. A normal behaviour here is the correct login attempts, while the abnormal behaviour is the incorrect login attempts. Many incorrect login attempts is a clear indication that something is going on. The pros listed in the paper are that it is effective to detect new vulnerabilities, less dependant on the operating system (an abnormal behaviour is mostly the same on both Windows and Linux systems for instance), and that it can see abuse of privilege (for instance an admin who does many things in a system he should not do). The cons listed in the paper are that observed events constantly change (a system with 5 nodes being expanded to 50 nodes will have different normal and abnormal behaviour for instance), that the system is unavailable while rebuilding behaviour profiles, and that it is difficult to trigger alerts at the right time.

The last category Liao [8] lists is stateful protocol analysis. Stateful protocol analysis is somewhat similar to behaviour based detection, but they are different. SPA focuses on unknown attacks, whereas behaviour-based detection focuses on known attacks. SPA can track the states on network connections. The pros with SPA listed in the paper are that SPA know and trace the protocol states and that SPA distinguish unexpected sequence of commands. The cons listed in the paper are that SPA is resource consuming, SPA is unable to inspect attacks looking like benign protocol behaviours and it might be incompatible to some operating systems.
Detection approaches

In the section above mostly detection methodologies were covered. Liao [8] also covers many detection approaches which the methodologies can use. The approaches listed in the paper are statistics based, pattern based, rule based, state based and heuristic based. The approaches are only covered briefly by the paper, but the paper does have a number of references of each approach. According to Debar [2] (a reference cited in Liao’s paper), the most common tool to build a behaviour based system is statistics. This makes sense because behaviour is measured over time. However Debar also states that a more complex statistical model has to be built, because the original model was too simple. The original model was based on averages and standard deviation which is a quite simple model. Many Machine Learning algorithms are also based on statistics, but that is covered later. A pattern based approach on the other hand focuses on string matches. [8] A typical pattern based approach is used to discover for instance SQL injection. According to Halfond [4] SQL injection is an attack which input data is classified as SQL code. By doing this an attacker can send raw SQL commands into a database and cause serious damage. This is a common attack in web applications and probably one of the most attempted. Another problem with some database platforms according to Halfond is the level of detail in error messages. An attacker can insert some kind of a known invalid SQL command and then make the system throw an error message. If the message is too detailed the message may tell something critical about the system encouraging the attacker to insert a valid SQL statement next time. The reason for briefly describing SQL injection attacks is that this is a very typical attack where a pattern based approach can be used to detect the attack. A SQL statement can look like this (just an example):

```
select * from customers where surname = 'DOE';
```

There is no doubt that this type of string is very common when dealing with SQL databases. When using a pattern based approach one may take a look at the input field where typically a common input is a username and a password. An uncommon pattern will, of course, be a SQL statement in these fields. A SQL statement in these fields should trigger an alarm in the IDS. The next detection approach in Liao [8] is rule based. "If-then and if-then-else" rules are applied to create a model and profile of known attacks. [8]. Actually one of the areas where Liao mentions a Machine Learning algorithm is here. The algorithm mentioned is support vector machine. Machine Learning algorithms will be covered in more detail later on in the thesis, but since a paper on support vector machine is mentioned a short introduction will be made to this algorithm. The paper mentioned about SVM is Li. [7] Li also states that SVM is a very popular algorithm used for intrusion detection due to its robustness and efficiency. IDS deals with an enormous amount of data with much redundancy, so the paper also describes some methods to deal with that. [7] The conclusion in the paper is that the SVM algorithm did in fact predict 98.6249% correctly so
it certainly shows that this algorithm is powerful. More about this topic and algorithm is covered later. The next approach mentioned in [8] is state based methods. Liao says in the paper that state based methods use finite state machine to detect attacks. One can simply describe a finite state machine as a mathematical model with many different states. Wikipedia has some simple examples - for instance a turnstile gate at a metro station. The last approach mentioned in [8] is the heuristic based. This approach is based on artificial intelligence. This is also related to Machine Learning.

Types of technology

Liao [8] also mentions different types of technology, some of them which were briefly introduced earlier. In addition to host based IDS and network based IDS, Liao also introduces wireless based IDS, network behaviour analysis and mixed IDS. Host based IDS and network based IDS are both covered earlier on, so this section will focus more on the other technologies. Wireless based IDS is similar to network based IDS but the focus is now on wireless traffic. One can ask for the need of this technology, since wireless networks are usually connected to a cabled network. There are, however, attacks that are specifically intended for wireless networks, like listening to the wireless traffic. This is not easy if you only have a cabled network to deal with. A wireless based IDS should be able to detect these types of attacks. A network behaviour analysis is also mentioned in the paper. The paper states that NBA inspects network traffic to detect attacks through unexpected traffic flows. A normal day at the office one may expect many http packages to flow through, many smtp (e-mail) and other kind of communication packages. However one may not expect a lot of ICMP packages (PING). Many ICMP packages may be a sign of a denial of service attack. A denial of service attack is simply when a node or service is overflown with packages. The most common way to do this is to send a lot of ICMP packages. Now preventing a DoS attack based on ICMP is easy, but there are other ways of doing a DoS attack. There was for instance a vulnerability in the blogging software Wordpress which enabled easy DoS attacks. The last technology mentioned in the paper is mixed IDS. This technology as one can probably read out of the words mixes different technologies. A mixed IDS can be very powerful as the technology can take advantage of the strength of each technology. However what might be a reason for companies not to install a mixed IDS is the cost. The cost of installing might be too high and may also be costly to update with new attacks.

Virtual machines

Liao [8] also mentions virtual machines. Liao defines virtual machines as a software implementation that emulates the functionality of a real machine. Today many use virtual servers on one physical server. One of the most important benefits is that one can store many servers on a server. One can even create own virtual networks for the machines. The main benefit with that is that an attack against one virtual machine does not need to
affect others. However an attack against the physical infrastructure might be catastrophic. It is therefore very important that one tries to "hide" the physical infrastructure away from the virtual networks. Another concern which Liao mentions about virtual machines is that virtual machines easily can be copied. It is very easy to copy a VM so that one does not need to configure everything, however if that VM is full of security holes that will apply to the newly copied VM. One could imagine if there is one network with many exploited VMs and one has email, webserver and for instance a file server. The file server might have been copied from the webserver and that might have security issues. The security issues will then remain in both the servers. However, Liao also addresses potential solutions to this problem. By adding traditional IDS techniques to each VM one can prevent many attacks. One can even assign one VM to monitor the rest of the VMs. To have proper IDS on virtual machines is important because this is also related to cloud computing.

Cloud computing

This is also an interesting topic when it comes to intrusion detection and is also related to virtual machines. Storing data in the cloud is becoming more and more popular. Services like Dropbox and One Drive are gaining popularity due to the fact it is a simple way to store data without storing it on a physical drive which could become damaged. However with this more and more people store data they cannot afford to lose and many of these data are potentially sensitive. It is for instance much more handy to store a copy of your doctor’s prescription on the cloud than having it on a paper which you can lose. With more and more users, this requires more and more security. Zissiz and Lekkas [21] introduce three service models of cloud computing. These are Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). IaaS according to Zissiz and Lekkas provides the consumer with full control of operating system, network, and storage. This is as one can see related to virtual machines which is reviewed in the section above. PaaS according to Zissis and Lekkas provides the customer with the abilltiy to deploy programs and other services, but the customer does not on the other hand control anything about storage and operating systems. SaaS is probably the simplest of the three models with the user only controlling a limited amount of user specified settings. A common usage of this service is thin clients at work for instance or when you use a Citrix service to log in remotely. Zissis and Lekkas [21] furthermore introduce four deployment models of cloud computing. These are private cloud, community cloud, public cloud and hybrid cloud. What is important to have in mind is that these are how the cloud infrastructure is operated. These deployment models are quite simple according Zissis and Lekkas. A private cloud is operated for a private company, but might be operated by a third party. A community cloud is where the cloud infrastructure is shared between several organizations.

1CITRIX is a very common SaaS software used by many, more information: https://www.citrix.com/
with more or less the same needs. A public cloud is where the cloud infrastructure is made available to the general public. An example of this is Amazon AWS. One paper by Somorovsky et al specifically deals with vulnerabilities when it comes to the Amazon cloud. In this paper vulnerabilities with the web interface are discovered. Originally Amazon was a bookshop on the Internet, but nowadays Amazon offers a wide array of products and services. The login interface to the AWS is the same as one uses to login into the web store. Somorovsky then uses a XSS vulnerability to gain access to the control panel of virtual machines. One thing is to gain control over one single virtual machine, something else is to gain control over the control panel where one has complete access to close and delete instances. Amazon AWS uses a SOAP based interface. Somorovsky then discovered vulnerabilities in the SOAP based interface.

2.2 Review of Machine Learning

One can find the following definition of Machine Learning on Wikipedia: "Machine Learning is a subfield of computer science that evolved from the study of pattern recognition and computational learning theory in artificial intelligence." As mentioned in the introduction one of the most common uses of Machine Learning are spam filters on your email. When you log into your Gmail account you will notice you have a folder called spam (or in Norwegian søppelpost). In the early days of email spam filters were not there so you could end up spending a lot of time deleting all the spam. Gmail is a particular interesting case to analyse. They even have folders for advertisements (e.g. campaigns for cheap airline tickets), social media (e.g. email notifications from Facebook, Twitter etc) and of course your primary folder where all important email (or what Gmail thinks is important to be clear) are located. All these folders are in addition to the spam folder, where all the real junk go (e.g. letters from Nigeria).

So how does Gmail knows all this? One could imagine a situation where Gmail had to employ people to go through each individual email sent into the Gmail servers and place them in the right folder. If they had to do this there would be work for every single individual on the planet. The idea is good, but not practical. The way Gmail knows this is of course by different Machine Learning algorithms. By using these algorithms Gmail can look for patterns in each email and direct the email to the relevant folder.

2.2.1 Usage of Machine Learning

I have already provided an example of how Machine Learning is used in email. I will in this section take a closer look of usage of Machine Learning

---

2Amazon Web Services
3Cross site scripting - a vulnerability that allows attackers to use client-side scripts into websites
4Simple Object Access Protocol
today. Machine Learning Mastery [9] has a list of 10 examples of Machine Learning problems. One interesting example which is in that list is medical diagnostics. Medical professionals can by inserting a list of different symptoms into a software, get a prediction on whether the patient is ill or not. However, if a patient obtains access to these systems we may have a lot of self diagnosis of illnesses. There are already many websites which do this with lists of symptoms.

Another usage according to Machine Learning Mastery is speech understanding. One example is the iPhone Siri. [20]. Siri can help you to find your way or just find the nearest restaurant from the location you are at. Siri was not, however, Apple’s original idea. It was first released in the appstore of Apple by another company before it was bought by Apple. The funny thing though is that Google’s own voice search for iPhone was found to work better than Siri. Speech understanding can pose some problems. Siri has been widely criticized for not understanding certain accents. These are just two examples of Machine Learning Mastery used today and we see that the number of applications using Machine Learning Mastery will undoubtedly increase in the years to come. In the next chapter, techniques used in Machine Learning Mastery will be discussed.

2.2.2 Supervised techniques

One can classify different Machine Learning algorithms into different techniques. This section will take a closer look at supervised techniques. An algorithm using a supervised technique contains both the input and the result. This is done to create a model. Machine Learning Mastery simplifies this to $y = f(x)$ where $y$ is the output variable and $f(x)$ is the input variable. The goal here is to predict the output data from the mapping function. Two examples of where supervised techniques are used are classification and regression. Classification is where the output is a category such as red or blue whereas regression is where the output is a real value, for instance how many dollars you have. The key here is that the output is predicted from the input data. There are some issues of course with supervised techniques. According to Kotsiantis [6] some of the main issues with supervised Machine Learning techniques are impossible values have been input, no values have been input and irrelevant features are in the data. Another challenge according to Kotsiantis is that the data is incomplete, but that is unavoidable in the real world. Research will then have to handle these problems. Kotsiantis describes three methods of doing that. One can leave the feature unknown, one can leave the feature so it does not exist and one can ignore the given feature. The choice of which algorithm to be tested is also important when dealing with incomplete data. Kotsiantis also present a wide array of different supervised techniques, which will now be covered. The different techniques are logic based algorithms, perceptron based techniques and statistical learning algorithms.
Logic based algorithms

In this category decision trees are found. Decision trees are quite simple. One node of the decision tree represents an input, whereas the edges in the tree represent possible input variables of that node. Decisions trees have the advantages that they are easy to visualize and therefore also easy to understand for beginners in Machine Learning. Quinlan [11] provides a very good example of how to use a decision tree. The paper is from 1986, but the principles of decisions trees have not changed since then. The example in Quinlan’s paper is Saturday mornings as a object and the attributes are outlook with the values of sunny, overcast and rain, temperature with the values cool, mild, hot, humidity with the values high and normal, and windy with the values true and false. One can already see that this is suitable for a data set. An example provided for one particular Saturday morning in this paper is outlook - overcast, temperature - cool, humidity - normal and windy - false. As the concept of Machine Learning is usually pattern recognition one can see that by collecting this data they can tell us something a pattern when it comes to weather on a Saturday morning. A challenge with a decision tree is that each individual has to find individual parameters to start with. This may make the decision tree quite inaccurate when it comes to objects outside the training set so it is therefore important to find the parameters which cover the most. Quinlan’s paper covers this issue.

Perceptron based techniques

In this category neural networks are found. According to Kotsiantis [6] the most common way to learn a perceptron based algorithm is to run through the algorithm on the training set repeatedly so that it finds a vector which is correct on all of the training set. One important feature with perceptron based techniques is that they are binary (output either 1 or 0) so if dealing with a non-binary problem one must reduce the classes to multiple binary problems. One usage of neural networks which is particular interesting is face detection in photos. This is a real world problem which is useful to for instance law enforcement. Rowley et a.l [12] performed this in their paper. What they did was to have two outputs - 1 for facial examples and - 1 for non face examples (remember the binary nature). With these two outputs they had a 20X20 pixel filter from each image so they could easily train the algorithm to detect facial and non facial areas. The filter was applied to both facial and non facial areas of the images. They also trained the algorithm on non face images (for instance scenery). The conclusion of this paper was a detection rate of between 77,9% and 90,3% of the facial images. The main limitation of this work was that the algorithm could only discover faces looking directly on the camera. Another common usage of neural networks is found in medicine in particular.
Statistical learning algorithms

The most important feature of these algorithms are that they have a underlying probability model. The most important and well-known algorithms in this category according to Kotsiantis [6] are the Bayesian networks. Kotsiantis describes naive Bayesian network more specifically and Bayesian networks more in general. A naive Bayesian network is a more simplified form of a more general Bayesian network. The main advantage of the naive Bayesian network according to Kotsiantis is the short computational time of training. However a Bayesian network uses a product to determine probability so a probability close to 0 might cause some problems. There are ways to avoid it described in the paper. So where do you commonly use Bayesian networks? The nature of a Bayesian network is dealing with probability. A simple question what is the probability of person driving under influence of alcohol? The probability of a person doing that with zero alcohol intake is close to zero. The risk is first there when a person has some alcohol intake. A Bayesian network may then been drawn up to illustrate the probability.

Instance based learning

Instance based learning algorithms are closely related to statistical learning algorithms. Kotsiantis [6] references to a work from as early as 1967 to describe the basic properties of the KNN algorithm. The description is "k-Nearest Neighbour (kNN) is based on the principle that the instances within a dataset will generally exist in close proximity to other instances that have similar properties". A pseudocode of the algorithm also from Kosiantis [6] is as following:

```
procedure InstanceBaseLearner(Testing Instances)
for each testing instance
{
  find the k most nearest instances of the training set according to a distance metric
  Resulting Class= most frequent class label of the k nearest instances
}
```

The pseudocode makes it easier to see what the algorithm actually does. One has both training and testing instances. The distance metric here is the value of k. The result is the most frequent labels according to the value of k. The main disadvantage of this algorithm is to find the optimal value of k. The larger the value of k, the better the algorithm predicts. However, setting a very large value of k will make the algorithm very costly to run. The K-nearest neighbour or KNN algorithm is the main topic of this thesis and will be discussed further in chapter 3.
2.2.3 Unsupervised techniques

According to Machine Learning Mastery unsupervised techniques are techniques where there is only input data, but no corresponding output. These algorithms focus more on working out a underlying structure or distribution of the data. Typical problems described on Machine Learning Mastery are clustering and association. Clustering is used where one would like for instance to group customers by purchase behaviour where as an association describes, for instance, people who buy X also tend to buy Y. With these tools one can get useful data about customers for instance. Jain et al has a general review paper which discusses different clustering techniques and it contains some interesting issues [5]. One of the biggest challenges with clustering is subjectivity. This means that the same set of data must be partitioned differently for different applications. Two algorithms in this category are k means and mixture problems.

K means algorithm

One of the most popular unsupervised algorithms used is the K means algorithm. K means clustering means that one divides observations into a k number of clusters. This algorithm should not be confused with the KNN algorithm which is a supervised algorithm and not a unsupervised algorithm. In fact they are not similar at all. K is just a random constant in both cases. The main advantage with K means is that the algorithm is low in cost. The algorithm is also quite simple.

2.2.4 Machine Learning usage in IDS

Machine Learning is not widely used in IDS today, despite the fact that the area is heavily researched. According to Sabhani and Serphen [13], the algorithms were not that effective to discover user to root and remote to local attacks. A user to root attack is an attack where a user has some access to a system and then tries to gain root access (e.g. by exploiting flaws in an OS or by social engineering). A remote to local attack is of course when somebody outside the network tries to attack your system. These are two major flaws in the Machine Learning algorithms that were tested in the paper above. This reason alone may be one big reason for why Machine Learning is not widely used. The algorithms must be able to discover such basic attack types. A good example of a remote to local attack is when an attacker takes control of your system to use it in a botnet. Now we have to take in account that the paper mentioned is from 2003, so normally the algorithms should be improved by now. The dataset used (KDD CUP 1999)) though is still heavily used in research, however, there are some criticism of it that it does not resemble real network traffic. I will use the KDD Cup dataset later on to the thesis.

Barreno et al [1] takes another approach. This paper asks if Machine Learning algorithms are secure. This is of course a relevant question to
The paper provides some very specific problem statements. The first question asked is if an attacker can manipulate an algorithm to permit specific attacks. There is no doubt that manipulating a Machine Learning system to permit attacks will have serious consequences. The algorithms mentioned in this thesis are quite well known how they work. However, the system where the algorithms are set up may be less known. The next problem stated in the paper is that an attacker can confuse the system so that the administrator has to disable it. This can be for instance make the system reject proper email or traffic. The general question asked in this paper is if an attacker can exploit weaknesses in the Machine Learning system to exploit it. Barreno et al [1] also widens the context as Machine Learning techniques are not only used in email spam filters or intrusion detection but also many web services, virus detection etc. The paper also mentions the techniques (supervised and unsupervised), but since these are mentioned earlier, there is no need to go through them again. One important concept in the paper is the f(x) when it comes to intrusion detection. \[ F(x) = \text{normal or } f(x) \text{ intrusion} \] is a simple concept to understand. The concept of attack in this paper is also extremely important to understand. An attack in this paper is an attack on the learning system and not what one usually thinks of an attack. In other words an attack is to lead f(x) to misclassify attacks. The paper also mentions some attack models. These models are influence, specified and security. Each of these have some submodels. The submodels under influence are causative and exploratory. A causative attack is when one alters the training process. An exploratory does not alter the training process, but instead uses techniques to discover information. The submodels under specify are targeted and indiscriminate. A targeted attack is focused on a small set or a particular point. An indiscriminate attack focuses on more general points, for instance all false-negatives. The submodels under security are integrity and availability. An integrity attack is according to the paper an attack where intrusion points is defined as normal (false negatives). An availability attack is broader than an integrity attack. An availability attack yields so many errors so the system becomes unusable. This paper was one of the most interesting because most of the other papers only look at security of a system, not the intrusion detection system itself.

2.3 Confusion matrix

A confusion matrix according to Wikipedia is a table layout which visualises performance of an algorithm. [16] A confusion matrix makes it much easier to see if an algorithm predicted correctly. An example of this is below [16].

In this example from Wikipedia there were 8 cats, 6 dogs and 13 rabbits which means a total in 27 animals. Of the 8 cats this algorithm (what algorithm this is, is not mentioned, but it is not important) predicted 5 cats and 3 dogs. Of the 6 dogs the prediction was 2 cats, 3 dogs and 1 rabbit. The most correct prediction here was that of the 13 rabbits, the prediction was 11 rabbits and 2 dogs. This matrix can be further subdivided into a
### Table 2.1: Example confusion matrix

<table>
<thead>
<tr>
<th>Predicted class</th>
<th>Cat</th>
<th>Dog</th>
<th>Rabbit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cat</td>
<td>5</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Dog</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Rabbit</td>
<td>0</td>
<td>2</td>
<td>11</td>
</tr>
</tbody>
</table>

A confusion matrix is a table that shows the compliance between two data sets, such as a test data set and the correct one. Observations on the diagonal of the matrix are correctly classified, while observations in the other cells are erroneously included or excluded. Such matrices or cross tables are used in some of the experiments in chapter 4.

Recall is defined as the fraction of relevant items that are retrieved or [19], in other words, how many relevant items are selected. This means that if all items are found by the test, then the recall value is 100%. If some items are not found, then the recall value is less than 100%. Precision is defined as the fraction of retrieved items that are relevant or, in other words, how many selected items that are relevant. Precision will be less than 100% if too many items are identified in the test for a class.

The following terms are relevant for precision and recall:

- **True positive (TP)** is where an item is correctly retrieved.
- **False positive (FP)** is where there is a false alarm or where an item has been retrieved when it should not have been.
- **False negative (FN)** is where an item should have been retrieved but where it was not retrieved.

Precision and recall can then be calculated as:

- **Precision** = \( \frac{TP}{TP + FP} \)
- **Recall** = \( \frac{TP}{TP + FN} \)

Recall and precision are calculated for some of the experiments in chapter 4.
Chapter 3

Approach

The experiments will be performed on the Alto cloud provided by HiOA and on my own computer. The main advantage of doing experiments on virtual machines in the cloud, is that we can test in different environments (e.g. we can check performance with different amount of memory). One can also do experiments without having to worry too much about destroying anything. For collection of statistics I will use the programming language R [10]. The advantage of using R is that we can easily read data from large files and analyse them. With these two tools one can easily compare the performance between different algorithms and the more traditional IDS.

3.1 Types of Machine Learning algorithms

According to Machine Learning mastery there are two main categories of Machine Learning algorithms. The two categories are grouping by learning style and grouping by similarities (e.g. grouping similar animals). Learning style algorithms typically have some "learnt" interaction with the data we would like to interact with. This can for instance be that the algorithm has learnt to distinguish spam from not spam. Algorithms grouped by similarity, however, are grouped by how they work. In this category we find tree based algorithms and bayesian based algorithms. I will now go through some types in the different categories.

I will start with the algorithms grouped by learning style. One algorithm in this category is supervised learning. In this algorithm we have a known label (e.g. spam or not spam), and the model is then "trained" to learn what kind of data the label contains. Another model in this category is unsupervised learning. In this model input data is not labelled and we do not have a known result. Mathematical rules are used to organize and recognize the data. A third algorithm in this category is semi supervised learning. In this algorithm we have a combination of the two already mentioned (e.g. both labelled and unlabelled data). This algorithm is more flexible in terms that it has to handle both.
In the algorithms grouped by similarity we have more algorithms than the ones grouped by learning style so I will only cover the most important ones. One example here are the decision tree algorithms. These algorithms are one of the most popular ones when it comes to Machine Learning. It is quite simple to understand. The decision tree has a set of decisions based on the data attributions. For instance a packet can arrive with a label (e.g. ICMP) and then the algorithm will take a decision of what kind of action to do. These actions can for instance be to drop the packet, to let the packet pass or investigate the packet closer. In a decision tree this moves the packet down the tree based on the instructions the algorithm have.

There are also instance based algorithms. In this group we find for instance K-nearest neighbour. This is a distance based algorithm. The simplest way to describe this algorithm is to start of by describing a distance (e.g. k=5). This will now direct us to the five nearest neighbours. Since the nearest neighbour algorithm is determined by a "majority vote" we can say that if there are 2 apples and 3 pears (remember k=5) - the pears will be the outcome of our "search", simply because pears have more "votes" than apples in this example. Nearest neighbour algorithm is one of the most commonly used algorithms in Machine Learning. When using R ties are broken randomly (e.g. if apples and pears get the same amount of "votes"). This is of course a very simple description of this algorithm. There are of course different methods to measure the distance k. We can for instance give each neighbour a weight which counts more than a vote.

Another algorithm type which is more advanced than the clustering and the classification algorithms are the artificial neural network algorithms. This family of algorithms is in fact inspired by the central nervous system in humans. Real life applications of these algorithms include robotics, game-playing (for instance chess) and vehicle control. An artificial neural network has different nodes which are connected to each other.

### 3.2 Traditional IDS/IPS implementation

How is IDS/IPS implemented today? In most cases these are implemented on a server or in large corporations many servers to make sure every single part of the network is covered. As mentioned above SNORT is one of the most famous IDS. To set up SNORT one just needs a Linux host and then you can just start to look for intrusions if it set up properly with a set of rules. One can also set up your own rules in SNORT which is quite easy. Below is a sample rule in SNORT.

```plaintext
alert tcp any any <> 10.5.70.91 any (msg:"All traffic yoho"; sid:1; rev:3;)
```

This rule is very simple and actually not very useful as it does generate alerts for ALL TCP traffic in and out out of the IP address 10.5.70.91. The rule is, however, very useful for testing purposes to make sure SNORT
works properly. One also has other options than alert. One can for in-
stance replace alert with the log parameter. That means that we actually log
the traffic specified. SNORT can of course run as a standalone installation
(console mode), but one usually has some sort of web interface to make
managing SNORT much easier (and also easier to read alerts and logs).

As mentioned earlier SNORT needs a set of rules or signatures to make
sure we are aware of the newest attacks. SNORT rules come in different
sets. The free of charge community rules and the paid subscription rules. The
paid subscription rules are newer than the community rules, but usually the
SNORT developers release the subscription rules to the community rules
after a while. The subscription rules are also updated on a daily basis
more or less, making these rules safer to use. For a business the cost of
a subscription is 399 USD pr sensor/pr year. For personal use the price
becomes much cheaper, but this is almost always the model in licensing
costs.

3.2.1 SNORT ruleset

Luckily the SNORT developers have released information about what you
get in the subscription rule set. There are quite many of them, but with
the number of vulnerabilities in a network, this is of course essential. One
interesting aspect with SNORT is that rules are not deleted permanently,
just moved to the category called deleted. The reason for this may be that
if there is an issue with a new rule, the old rule can easily replace the
newer one if needed. As mentioned above there is a lot of rule sets ranging
from rules about web browsers to rules about operating systems. The
SNORT developers have sorted the rule set quite logically which makes
it easy to know what to look for. All rules regarding a web browser
begins with browser-XXX and for operating systems os-XXX. There are even
rules regarding different protocols (for instance IMAP - which is an email
protocol). The subscription rules do actually have the community rules
included in them.

3.3 Technical part

As mentioned earlier on in this thesis, the technical part will be done
on ALTO cloud and also my own computer. R studio runs easier on
Windows computers than in the cloud where I only have a command line
interface. My computer has an Intel I7 processor, 8 gigabytes of ram and
a 244 gigabytes of disk space. In the ALTO cloud we can choose from
different configurations. In this thesis the experiments will run on the large
configuration. This configuration has 4 virtual CPUs, 8 gigabytes of ram
and 80 gigabytes of disk space. The producer of the virtual CPUs, however,
is not mentioned, but the configuration should be more than enough for
our use.
3.3.1 KDD cup dataset

The KDD cup dataset is a dataset which was used in a conference back in 1999. To generate this dataset they used a program called TCPdump. TCPdump is a program which captures traffic on the network. This thesis will use the full dataset which is 743 megabytes uncompressed. The dataset has over 4 million entries which gives unlimited amount of opportunities to apply different Machine Learning algorithms and then analyse the results. As mentioned earlier the data is already labelled which means that we do not need to do much before we run different algorithms on the set. Since it is labelled we know if a packet is “normal” or if there is an attack going on.

One can now take a closer look at the attack types described in the KDD cup dataset. There are four main categories in the dataset. These are DOS (Denial of Service), U2R (User to Root), R2L (Remote to Local) and PROBE(probe attack). A DOS attack is when an attacker makes the machine too busy to handle any legitimate requests. This type of attack is very common, but it is also easy to prevent. Many attackers who perform this attack use several machines in a botnet to make the attack more powerful than it would otherwise be. The KDD cup dataset contains six different DOS attacks. These are SMURF, NEPTUNE, POD, TEARDROP, LAND and BACK. A U2R attack is when an attacker has a normal user account and then uses some exploits to gain root access to the system. The dataset contains four attacks of this type. These are buffer-overflow, loadmodule, rootkit and perl. A R2L attack is when an attacker tries to gain access via the network on a machine the attacker does not have an account on. The dataset contains 8 attacks of this type. These attacks are warezclient, multihop, ftp-write, imap, guess-passwd, warezmaster, spy and phf. A PROBE attack is when an attacker gathers information for the purpose of finding exploits. The dataset contains four attacks of this type. These are nmap, satan, portsweep and ipsweep. The attacks mentioned here are all subtypes of the main category, and it is probably enough to know the main categories. Of these categories DOS and PROBE attacks are the most common ones, the two others are in fact much rarer. DOS is, however, much more common than PROBE again. It is, however, not difficult to understand that nmap typically involves using the program NMAP to scan for information.

The KDD cup dataset also of course shows the protocols in the dataset. These are TCP, UDP and ICMP. The TCP protocol is probably the most important one. With TCP we do get reliable transfer, and hence this protocol is used in for instance SMTP(email) and HTTP(websites) The UDP does not have reliable transfer and this protocol is then used for instance in transfer of SKYPE phone calls. The reason for this is that the UDP protocol is much faster. TCP will then require much more time to make sure the packet has arrived. The last protocol in the dataset ICMP is mostly used to send messages over a network.

The contents of the KDD cup dataset is quite interesting. The training
set (which is 10% extracted from the original dataset) has about 80% connections related to an attack and just about 20% normal connections. With these numbers in mind it should be quite easy to get an idea of how good different algorithms are to discover attacks. One can assume that the same numbers roughly correspond to the full set of data.

3.3.2 Performance of algorithms

Before one can perform the technical part one needs to have a process so that a plan can be laid out for all the experiments. In this way it will be easier to perform the experiments. So now to the very basic question - how good are the Machine Learning algorithms to discover attacks in the KDD CUP dataset? There are numerous papers on this. One paper is the Sabhani and Serpen. [13] That paper is from 2003 with less powerful hardware that is available today. The hardware they tested on was 400 mhz on the processor and 512 megabytes of RAM. The conclusion of this paper is that actually none of the algorithms they tested could discover U2R or R2L attacks in a significant way. Actually they were not able with any algorithms to discover any more than 30 % of the U2R attacks and any more than 10 % of L2L attacks. Also there were some algorithms that were better than discovering some types of attacks than others. This makes it of course more challenging to determine if there is one algorithm which is the best algorithm.

Now while algorithms themselves do not change over time, the power of hardware does. This might be something to consider when doing tests. Are the algorithms more powerful with more power than the original coding of the algorithms? Or do you only use less time to analyse something? As a CPU only provides power to perform tasks faster, the algorithm itself is still programmed the same way. A plausible answer to the question should be no. As a conclusion, a fast CPU does not affect the way an algorithm is programmed, but rather gives us better performance. In my case I may not need to wait several days to obtain a result from the 4 million entries of data. This is of course an huge advantage of CPU power.

There are of course many algorithms out there to test. R has built in many Machine Learning algorithms in a way so one does not need to program them in another programming language. With this the results are generated quickly and there is no need of programming these algorithms. Importing a dataset in R is also quite straightforward. In the screenshot below the interface of R-studio is seen. In this screenshot one can see how large the dataset actually is. So how does one perform a K-nearest neighbour algorithm in R? In fact R has
this implemented if you download some packages. Here is a closer look at
the code required for this.

```r
knn(train, test, cl, k = 1, l = 0, prob = FALSE, use.all = TRUE)
```

This is the syntax R uses to apply the KNN algorithm. Now a little bit
about the parameters. The parameter `train` is a matrix or data frames of
training set cases. The `test` parameter is a matrix or frame of testing set
cases. The `cl` is a factor of true classifications of the training set. The `k=1`
parameter is the value of `k`, which is how many neighbours which will be
considered. The `l` parameter defines the minimum number of votes for
a definite decision which in this case is 0. The `probe` is a parameter which
returns the proportion of the votes required as the winning vote if it is set to
true. In this example the parameter is false. The `use.all` parameter controls
the handling of ties.

### 3.3.3 R studio

Now a little bit closer look on R studio with the KDD dataset actually
imported. Here we can see the imported dataset.

As one can see here the dataset has nearly 5 million entries. The dataset
has 42 variables, which in Machine Learning can translate into features.
However, it has to be said that most of the variables are integers and may
therefore be of less interest to analyse. Now one can take a look at how
many packages there are of each type in the dataset.

```r
table(kddcup.data$V2)
```

<table>
<thead>
<tr>
<th></th>
<th>icmp</th>
<th>tcp</th>
<th>udp</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>2833545</td>
<td>1870598</td>
<td>194288</td>
</tr>
</tbody>
</table>

By using the `table` function in R makes this easy. The `$V2` parameter is the
second variable of the dataset. Some examples are here:

- `$ V1 : int 0 0 0 0 0 0 0 0 0 0 ...
- `$ V2 : Factor w/ 3 levels "icmp","tcp","udp": 2 2 2 2 2 2 2 2 2 2 ...
- `$ V3 : Factor w/ 70 levels "aol","auth","bgp",...: 22 22 22 22 22 22 22 22 22 22 ...
- `$ V4 : Factor w/ 11 levels "OTH","REJ","RSTO",...: 10 10 10 10 10 10 10 10 10 10

The reason for why we use the `table` function on the `$V2` variable is that this
variable contains `icmp`, `TCP` and `UDP`. So what does this observation tell
us? As one can see the protocol with most entries in the dataset is `ICMP`. A
wild guess may imply that most attacks in the dataset are of the type `DOS`.
One can also take a look at the $V3$ variable. From the code above this may imply that the types of connections are stored in this variable. One can take a closer look at some of the connection types.

```
http
623091

private
1100831
```

From this one can see there are 623091 http connections in the dataset and 1100831 private connections. There are of course many more connection types in the document, but due to space constraints in the document, it is impossible to view all of them. There is absolutely no suprise that there are many http connections in the dataset. Every time a website is loaded many http packets are sent back and forth. There were even some http packets on port 443, which is the port used for https traffic.

There is also one more important variable here. The $V42$ variable contains the name of the attacks.

```
table(kddcup.data$V42)
( . . )
smurf 2807886
```

Again due to space constraints it is not possible to view the whole output, but smurf is the one attack with more numbers than anything else. This is no suprise at all. As mentioned earlier smurf is a DOS attack and there were clearly most ICMP connections in the dataset. On the other hand there were quite few normal connections compared to the number of attacks. There were just 972781 normal entries in the dataset. With knowledge of these numbers we know what to look for when applying Machine Learning algorithms to the dataset. R has of course many more features than this, this is just a brief introduction of R with an imported dataset.

**Performance**

Some of the experiments will also be tested with time. In R this is easy with these lines of codes

```
1
2 system.time({
3     Do something that takes time
4     x <- 1:100000
5     for (i in seq_along(x)) x[i] <- x[i]+1
7 })
```

1Script from: http://www.cookbook-r.com/Scripts_and_functions/Measuring_elapsed_time/
3.3.4 Algorithms to be tested

The KNN algorithm has been mentioned several times, and is of course a central algorithm in Machine Learning. There are, however, many more algorithms to test. A good starting point is to test at least three algorithms, with KNN to be one of them. Serpen and Sabhnani [13] tested 9 algorithms. However, the KNN algorithm was not one of them. Instead they focused more on clustering algorithms. It is very important to know that the KNN algorithm uses a completely different method than the K-means algorithm and that the letter \( k \) in both algorithms is just a coincidence. Clustering algorithms organize the data into clusters of where the data has the most in common where as KNN is more of a classification algorithm. In learning style - the KNN is supervised where as K-means is unsupervised. So to begin with it makes sense to test them both on this dataset. This will clearly give some results that may be quite different to each other.
Chapter 4

Experiments

In this chapter all the experiments will be presented. As mentioned in chapter 3 at least three algorithms will be tested on the KDD cup dataset. Then an experiment with one of the algorithms using SNORT will be presented. An analysis and a detailed discussion of the experiments will follow in chapter 5.

4.1 Experiment 1

Task: To analyse the KDD cup dataset using the KNN algorithm. Tools to be used are the dataset and R-studio. The goal is to see how well the KNN algorithm can discover attacks in the KDD cup dataset. The background information of the dataset is given in chapter 3.

4.1.1 Steps

To perform this experiment we need to follow some steps. The first step is of course to load the data into R-studio. Training of the algorithm is also needed so one has to define labels so the algorithm can be properly trained to recognize the attack. Then after the training part one can use the algorithm on the whole dataset. The steps in the experiment can be summarized as follows: Load data -> Train algorithm on small dataset -> Run algorithm on complete data set -> Analyze results. These steps are used in all the experiments.

4.1.2 Preparing R

To make the KNN algorithm to work we need a package called class. To use this package we need to type\footnote{Note: The source code in this experiment is taken from https://www.datacamp.com/community/tutorials/machine-learning-in-r}

\begin{verbatim}
library(class)
Warning message:
package ‘class’ was built under R version 3.2.4
\end{verbatim}
This function tells us that the package is installed, but that the package may be a bit old. Note - this package is not installed by default in R, but it was installed before performing this experiment. So the next step is to prepare the data. If this data was not labelled there This means that one feature will not be overemphazised compared to another and the projections will be more accurate. To normalize a custom function has to be created. The source code for this can be seen below

```r
normalize <- function(x) {
  num <- x - min(x)
  denom <- max(x) - min(x)
  return (num/denom)
}
```

However, since our dataset is already labelled there is no need to normalize the dataset. The algorithm can then be applied to the dataset, but before that can be done a sample from the dataset needs to be randomly chosen (2/3 of the dataset will act as the training dataset).

```r
set.seed(1234)
nd <- sample(2, nrow(kddcup.data), replace=TRUE, prob=c(0.67, 0.33))
```

The `set.seed()` function is the random number generator in R. The `sample()` function in R gives us a sample of the data. In this case the number 2 is to assign either 1 or 2 to all elements in our dataset, `replace=true` means that that after assigning 1 or 2 to a vector, the next vector will be reset Now after preparing this, it is very important that the data is categorized between the test data and the training data. However, since the KDD cup data fails this experiment will instead use the IRIS dataset which is built into R. The reason for this failure is that somewhere in the dataset something is divided by zero. The steps are about the same as demonstrated above, but with fewer entries. The results, however, should be about the same. The IRIS dataset covers the SEP AL species of flowers. This dataset has only 150 entries, so the predictions may of course be more accurate than predictions on the KDD dataset. Now a closer look at how the IRIS dataset looks like.

<table>
<thead>
<tr>
<th></th>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
<th>Petal.Length</th>
<th>Petal.Width</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.1</td>
<td>3.5</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>2</td>
<td>4.9</td>
<td>3.0</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>3</td>
<td>4.7</td>
<td>3.2</td>
<td>1.3</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>4</td>
<td>4.6</td>
<td>3.1</td>
<td>1.5</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>5</td>
<td>5.0</td>
<td>3.6</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>6</td>
<td>5.4</td>
<td>3.9</td>
<td>1.7</td>
<td>0.4</td>
<td>setosa</td>
</tr>
</tbody>
</table>

These are the 6 first entries in the dataset, the dataset is sorted so the other species do not come before well into the dataset, so here is a sample from the other species.
To make this dataset suitable to run the KNN algorithm on, the entries have to be sorted randomly. There are 50 of each type of species, so the predictions of the species may be less accurate than what would have been the case of the whole KDD cup dataset. All source code for the experiments are in the appendix chapter except for the results. Then the training set, test set and labels are defined:

```r
> iris.training <- iris[iris_sample==1,1:4]
> iris.test <- iris[iris_sample==2,1:4]
> iris.trainlabel<-iris[iris_sample==1,5]
> iris.testlabel<-iris[iris_sample==2,5]
```

The task in this experiment is to predict the species in the dataset so the training labels and the test labels are set from row 5 in the dataset. The data is now prepared and the algorithm can be run with \(k=3\) for the first part:

```r
> iris_knn
[1] setosa  setosa  setosa  setosa  setosa  setosa  setosa  setosa
    setosa  setosa  setosa  setosa  setosa  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
    versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  virginica  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
   versicolor  versicolor  versicolor  versi
```
The two tables displayed compare the predictions versus the actual dataset. The predictions clearly show that there is one error by predicting one *virginica* too much. The algorithm should if it was perfect predict the species in the *iris.testlabel* variable.

### 4.1.3 Short analysis

As mentioned above there was one error in this prediction, so the model is actually quite accurate when it comes to predicting. The sample size, however, of this dataset is quite small compared to for instance the *KDD* dataset. Still it was one error. A guess could be that the error margin when running this algorithm on *KDD* dataset is very huge. Even if predicting 70% of the attacks and the normal traffic correctly in numbers that is still 1.2 million errors. One interesting thing with this algorithm is to see if there are more or less errors with a different size of the training and test set. That is covered in the next experiment.

### 4.2 Experiment 2

So what happens if the training data is reduced to approximately 50% of the data? Will the *KNN* model be more accurate or less? A hypothesis of this may be that the model becomes less accurate, simply due to less data. The source code for this experiment is shown in the appendix chapter.

One can now look at the results:

```r
> table (knn50)
knn50
  setosa versicolor virginica
      25     27     28
> table (iris.testlabel50)
iris.testlabel50
  setosa versicolor virginica
      25     30     25
```

This clearly has some of the same results from experiment number 1 were approximately \( \frac{2}{3} \) of the data was used as the training data.

### 4.2.1 Short analysis

The algorithm’s performance of predicting the species is actually quite ok. The prediction of the *Setosa* species was 100% correct, but the predictions of *Versicolor* and *Virginica* species were both slightly incorrect. Compared to experiment 1, the deviations are in the same direction but a little bigger. However, the relative difference is not very significant to tell much about the algorithm’s performance when you reduce the training dataset. A further reduction is required, and will be done in experiment number 3.
4.3 Experiment 3

In this experiment an even further reduction of the training set will be done. A slight worsening of the correct predictions were found in experiment 2 so the guess here is that the predictions will be much worse. A test with approximately 20 % of the original dataset should show some clearer indications. The source code is in the appendix chapter, but it is of course just some adjustments of some parameters. Now a closer look at the relative deviations.

```
> table(knn_20)
knn_20
  setosa versicolor virginica
  38   40   47
> table(iris.testLabel20)
iris.testLabel20
  setosa versicolor virginica
  38   42   45
```

The results are about the same as in the two other experiments performed here.

4.3.1 Short analysis

Three experiments have now been done with different training sets, and the results here clearly show that there is no evidence of a weaker prediction from the KNN algorithm when the training sample is reduced from 2/3 of the dataset, then to 50 % and then finally to 20%. This makes the algorithm quite robust in terms of predicting the correct results no matter the sample of the training data. One could of course have run one last experiment with by reducing the training data to 10%, but since the algorithm has proved to be quite robust, this is not needed. Instead it would have been more interesting to see if you adjust the value of \( k \) in the algorithm.

4.4 Experiment 4

The purpose of this experiment is to see how the value of \( k \) affects the results of the KNN algorithm. A valid hypothesis here is that the algorithm’s performance in a significant way will be affected. The reason for this hypothesis is that different values of \( k \) will affect the neighbours the algorithm picks and this may yield completely different results than just adjusting the training data. If the value \( k=1 \) is selected that means that only the closest neighbour will be considered. The value of \( K \) is usually an odd number so that the chance of having the same amount of votes when running the algorithm is eliminated.

```
> table(knn_k)
knn_k
  setosa versicolor virginica
```

29
Here the value of $k$ is set to 5.

### 4.4.1 Short analysis

The result is exactly the same as in experiment 3. By increasing $k$ more neighbours are covered in the testing scope. The predictions may then become more accurate. The result is as mentioned above the same as in experiment 3 and hence increasing $k$ may until a certain point have more accurate predictions. This experiment can be tweaked. In this experiment only 20% of the data was used as the training set, but what happens if you increase $k$ and use 2/3 set as the training set?

### 4.5 Experiment 5

This experiment will be mostly the same as in experiment 4, but with a different training sample (2/3) of the dataset. This will give a better idea how much the value of $k$ matters to predict the right species. First the results from experiment 4.

```
> table(knn_k)

knn_k
  setosa versicolor virginica
       38      40      47

> table(iris.testLabel20)
is. testLabel20
  setosa versicolor virginica
       38      42      45
```

The prediction here was two too many of the *virginica* species.

```
> table(knn_k_2_3)
knn_k_2_3
  setosa versicolor virginica
       11      21      16

> table(iris.testLabel)
is. testlabel
  setosa versicolor virginica
       11      21      16
```

As one can see the prediction was 100% correct.
4.5.1 Short analysis

The prediction in this case was 100% correct (no false positives or false negatives). Now this is interesting. It seems that increasing the value of \( k \) actually increases this algorithm’s accuracy. Still there must be a point where the value of \( k \) becomes too large (or too small) to make any sense when it comes to accurate predictions.

4.6 Experiment 6

From experiment 5 the conclusion was that the larger the number of \( k \) the more accurate the KNN algorithm’s prediction was. The value of \( k \) was 5. What happens if the value of \( k \) is set to 20? The results from experiment 4 is in the appendix section so it is easy to compare. A hypothesis here is that the results should be exactly the same as in experiment 4.

```
> table (knn_k_20)
knn_k_20
    setosa versicolor virginica
11     24      13

> table(iris.testlabel)
iris.testlabel
    setosa versicolor virginica
11     21      16
```

The prediction here had more errors than experiment 5 which predicted everything 100% correct.

4.6.1 Short analysis

With \( k = 20 \) the prediction was wrong. The hypothesis was wrong and the result of this experiment has a significant error in it. One can therefore conclude that a large value of \( k \) does not make the algorithm better to predict the correct results.

4.7 KNN with KDD cup

6 experiments using the IRIS dataset have now been done and this gives an idea how the KNN algorithm works. However, the most important part of this thesis is to measure the performance of some Machine Learning algorithms to detect attacks As mentioned earlier the data set which is going to be used is the KDD cup dataset. The dataset used is a little bit modified from the original dataset. The main difference between this modified version and the original version is that some rows of the dataset are converted into integers rather than the original text. The reason for this is that non numeric values yields some errors in R studio. This should, however, not have any effects on the results. As there are quite many
entries (nearly 5 million), a random sample of 100,000 will be used in the experiments. A sample of 100,000 should be large enough to give an idea of how good the KNN algorithm is to detect attacks.

### 4.7.1 Preparing KDD cup

The source code to extract a random sample of a dataset in R is shown below.

```r
kdd_sample <- kddcup.data [ sample ( nrow ( kddcup.data ), 100000 ), ]
```

The `sample` function in R is used to sample data from any dataset. This line takes 100,000 random lines from the number of rows in the dataset. It is very important to use the `sample` function correctly to receive the right sample as just small adjustments in the function gives a completely different sample. The experiments will be mostly the same as with the Iris set.

### 4.8 Experiment 7

First a recap on what to test. The tests are going to be on how well the KNN algorithm is to discover attacks. Different values of k will be used and also different samples of training data will be used as in experiment 1-6. In this experiment 2/3 of the data set will be used as training data, and 1/3 as the testing data.

Now as one can see here there are too many ties, which means the the KNN algorithm cannot choose between the ties. To get rid of this problem the dataset has to be normalized.

```r
> kdd_norm <- as.data.frame ( lapply ( kdd_sample [1:41], normalize ) )

function (x) {
  num <- x - min (x)
  denom <- max (x) - min (x)
  return (num/denom)
}
```

The function `normalize` normalizes the data. With a normalized dataset the data should have less noise. However, after normalizing the data, there are still too many ties. The way to make this work properly is to have a sample small enough. Setting $k=1$ should in theory eliminate the problem. However, another way to eliminate the problem is to just take a small sample of the data. With less data the risk of the KNN algorithm of yielding too many ties is greatly reduced. Since 100,000 is not a small enough number to eliminate this problem a much smaller sample with 10000 is used.
4.8.1 KDD with a smaller set

First a closer look at the sample being used.  

```r
> table(small1$V42)
buffer_overflow    normal
   2      9998
```

This is not a representative sample of the whole KDD dataset, but this works in R.. With an overweight of normal traffic and underweight of attacks it should be easy to have an idea of how well the KNN algorithm works to predict the correct results. This experiment is the same as in experiment 1, just with another dataset and much more data. Now a closer look at the results

```r
> table(data_pred)
data_pred
buffer_overflow    normal
    0      3279
> table(data.testLabels)
data.testLabels
buffer_overflow    normal
    1      3278
```

The algorithm predicted 0 buffer-overflow attacks, but there was one of them in the testing set. Now a closer look at the time used to perform this analysis (the time script in R is provided earlier in this chapter)

```
user  system elapsed
 1.22    0.00    1.23
```

The interesting parameter here is the user parameter. System time is measured when the system uses some resources to switch between tasks for instance and some other usages. User is the program one uses now, in this case R studio. One can see the prediction took 1.22 seconds which is not too bad.

4.8.2 Short analysis

As predicted this algorithm was quite accurate in predicting attacks and the normal traffic. The error was in predicting no attacks, but at the same time there was just one attack in the test data. Actually as mentioned above there were few attacks in this sample from the KDD set, but predicting just one error in 3279 samples is quite good. The time to perform the prediction was 1.22 seconds which can be said to be quite good. This will also be done in the later experiments.

---

2This is a highly modified sample made by Anis Yazidi
4.9 Experiment 8

The task here is to see what happens when the training set is reduced from 2/3 to 50%. In experiment 2 there were more errors which is of course no surprise since the training set is reduced. Now with a larger sample than the IRIS set, there should be more errors.

<table>
<thead>
<tr>
<th>data_pred</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer_overflow .</td>
</tr>
<tr>
<td>normal.</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>4990</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>data_testLabels</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer_overflow.</td>
</tr>
<tr>
<td>normal.</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>4989</td>
</tr>
</tbody>
</table>

Now a closer look at the time:

<table>
<thead>
<tr>
<th>user</th>
<th>system</th>
<th>elapsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.43</td>
<td>0.00</td>
<td>1.44</td>
</tr>
</tbody>
</table>

4.9.1 Short analysis

The training data is here decreased to 50% of the original dataset. In experiment 1 and 7 the training data was 2/3 of the original dataset. This time the prediction yielded just 1 wrong prediction. 50% is probably a high enough portion of training data to yield few errors. This is about the same results that was in experiment 2. One can also see that the time was longer than in experiment 7, maybe that has something to do with reduced training data? To make sure this is the case some more experiments have to be done. The next three experiments showed 1.50 seconds, 1.55 seconds and 1.55 seconds. Now of course to get a clear idea of the time one has to do this experiment again and again at least a couple of thousand times, but at least it gives some idea. Less training data means more guessing, and therefore more resources is needed from the CPU.

4.10 Experiment 9

This experiment will be similar to experiment 3 with 20% of the dataset as training and 80% of the data as testing. In experiment 3 this did not yield many errors at all, even though in theory it should be more difficult to predict the correct data.

<table>
<thead>
<tr>
<th>data_pred</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer_overflow .</td>
</tr>
<tr>
<td>normal.</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>8014</td>
</tr>
</tbody>
</table>
4.10.1 Short analysis

The algorithm predicted that everything was normal, but in fact there were two attacks that the algorithm did not notice. The results, however, were quite similar to the results in experiment 3. The conclusion here is the need for more observations of attacks. The time for this experiment is quite interesting. In both experiment 8 and 9 the time used was more than the time used in this experiment. This might lead to the assumption that the amount of training data is not a factor when determining the cost of the algorithm. However, it will be interesting to see in the next experiments where the numbers of features are increased if the time will increase significantly. However, the total amount of data is not increased due to the sample only containing 1000 entries.

4.11 General observations

The results are quite similar with both the IRIS dataset and the reduced KDD cup dataset. Not all labels in the KDD dataset are used making the sample quite biased. There is a clear overweight of normal traffic whereas in the full dataset the normal traffic is just around 20%. A sample with all the labels should be used, but since the sample with 100,000 yields too many ties in KNN a smaller sample has to be used. In the following experiments a sample with 1000 will be used and it will be representative for the full dataset.

4.12 Experiment 10

Similar to experiment 1, just with 1000 samples from the full KDD cup dataset. The predictions and the facts are in the two tables below.

<table>
<thead>
<tr>
<th>Smurf</th>
<th>Neptune</th>
<th>Normal</th>
</tr>
</thead>
<tbody>
<tr>
<td>180</td>
<td>75</td>
<td>71</td>
</tr>
</tbody>
</table>

and the time for this experiment to perform:

<table>
<thead>
<tr>
<th>user</th>
<th>system</th>
<th>elapsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
4.12.1 Short analysis

With 180 predicted SMURF attacks, 75 predicted neptune attacks and 71 predicted normal traffic compared to 181 SMURF attacks, 68 normal traffic, 72 neptune attacks, 2 loadmodule and 3 portsweep attacks one can see that the algorithm clearly considered some of the attacks as normal traffic. The algorithm even classified some attacks as wrong attacks. The critical part here is not that the algorithm classified some attacks as another type of attack, but that the algorithm considered some attacks as normal traffic. The error percentage is, however, not very high so the threat should not be significant.

4.13 Experiment 11

As 1000 entries of the KDD gives a representative idea one can now see what happens with 50% of the set acts as the training set and 50% as the testing set. This experiment is similar to experiment 2 and 8.

The predictions and the actual facts are in the two tables below.

<table>
<thead>
<tr>
<th>Smurf</th>
<th>Normal</th>
<th>Neptune</th>
<th>Warezmaster</th>
</tr>
</thead>
<tbody>
<tr>
<td>279</td>
<td>94</td>
<td>101</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.3: Experiment 11 prediction

<table>
<thead>
<tr>
<th>Smurf</th>
<th>Normal</th>
<th>Neptune</th>
<th>Portsweep</th>
<th>Satan</th>
</tr>
</thead>
<tbody>
<tr>
<td>279</td>
<td>92</td>
<td>99</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.4: Experiment 11 facts

4.13.1 Short analysis

Once more the KNN algorithm predicted just a little bit too much normal traffic. This is actually not something a company wants. The question here after these two experiments are what kind of attacks the KNN algorithm predicts as normal traffic. A table function in R actually shows this. The following table shows the confusion matrix for experiment 11:

```r
> CrossTable(x = data.testLabels, y = data_pred, prop.chisq=FALSE)
```

<table>
<thead>
<tr>
<th>Cell Contents</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

36
<table>
<thead>
<tr>
<th>N</th>
<th>N / Row Total</th>
<th>N / Col Total</th>
<th>N / Table Total</th>
</tr>
</thead>
</table>

Total Observations in Table: 475

<table>
<thead>
<tr>
<th>data.testLabels</th>
<th>neptune</th>
<th>normal</th>
<th>satan</th>
<th>smurf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>0.004</td>
<td>0.021</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>0.000</td>
<td>0.004</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
</tbody>
</table>

| 99   |         |        |       |       |
| 1.000 | 0.000 | 0.000 | 0.000 | |
| 0.208 | 0.000 | 0.000 | 0.000 | |
| 0.208 | 0.000 | 0.000 | 0.000 | |

| 92   |         |        |       |       |
| 0.000 | 1.000 | 0.000 | 0.000 | |
| 0.194 | 0.979 | 0.000 | 0.000 | |
| 0.000 | 0.194 | 0.000 | 0.000 | |

| 2   |         |        |       |       |
| 1.000 | 0.000 | 0.000 | 0.000 | |
| 0.004 | 0.020 | 0.000 | 0.000 | |
| 0.004 | 0.000 | 0.000 | 0.000 | |

| 1   |         |        |       |       |
| 0.000 | 0.000 | 1.000 | 0.000 | |
|       |         |        |       |       |

37
The values for precision and recall for this experiment are (note here precision and recall are given in percent):
Smurf – Precision: 100%, Recall: 100%.
Normal - Precision: 92/94 = 97.8%, Recall: 100%.
Neptune – Precision: 99/101 = 98.0%, Recall: 100%.

Even if there is a lot of detail in the table above, it tells us something interesting. It tells that there are two packets of predicted normal traffic instead of the ipsweep attack. An ipsweep attack is where a host sends icmp packets to various hosts just hoping for a reply. Not a serious mistake as an icmp packet isolated is something anybody can send to check for instance internet connectivity. The most important result is that the error rate for actual attacks is low, for instance the prediction of smurf was 100% correct which is a ddos attack.

### 4.14 Experiment 12

Now one can reduce the training data down to 20% and take a closer look at the errors in the predictions. The predictions and the facts are in the tables below.

<table>
<thead>
<tr>
<th></th>
<th>Smurf</th>
<th>Normal</th>
<th>Neptune</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>456</td>
<td>155</td>
<td>198</td>
</tr>
</tbody>
</table>

Table 4.5: Experiment 12 prediction
4.14.1 Short analysis

The prediction here was correctly predicting the SMURF attacks, but failed on the others. In fact the prediction actually predicted too few normal traffic which is in this case would lead to false positives. It also predicted too many Neptune attacks. One can take a closer look at the table to see where the predictions were wrong.

<table>
<thead>
<tr>
<th>data . testLabels</th>
<th>neptune</th>
<th>normal</th>
<th>smurf</th>
<th>Row Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>back</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.002</td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>0.013</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>0.002</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>ipsweep</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>0.006</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>0.001</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>neptune</td>
<td>182</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td></td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.225</td>
</tr>
<tr>
<td></td>
<td>0.919</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.225</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>normal</td>
<td>11</td>
<td>152</td>
<td>0</td>
<td>163</td>
</tr>
<tr>
<td></td>
<td>0.067</td>
<td>0.933</td>
<td>0.000</td>
<td>0.201</td>
</tr>
<tr>
<td></td>
<td>0.056</td>
<td>0.981</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.014</td>
<td>0.188</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>portsweep</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.002</td>
</tr>
<tr>
<td></td>
<td>0.010</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.002</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>satan</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.004</td>
</tr>
<tr>
<td></td>
<td>0.015</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.004</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>smurf</td>
<td>0</td>
<td>0</td>
<td>456</td>
<td>456</td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.564</td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>0.000</td>
<td>0.564</td>
<td></td>
</tr>
<tr>
<td>Column Total</td>
<td>198</td>
<td>155</td>
<td>456</td>
<td>809</td>
</tr>
<tr>
<td></td>
<td>0.245</td>
<td>0.192</td>
<td>0.564</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.6: Experiment 12 facts
The values for precision and recall for this experiment are:

- Smurf - Precision: 100%, Recall: 100%.
- Normal – Precision $152/155 = 98.1\%$, Recall $152/163 = 93.3\%$.
- Neptune - Precision $182/198 = 91.9\%$, Recall: 100%.

11 normal traffic packets were incorrectly defined as Neptune attacks, and also 2 portsweep and 3 satan were also incorrectly defined as the same. Also 1 ipsweep and 1 back attack were incorrectly defined as normal traffic. How serious is this? A NEPTUNE attack is the same as SYN-Flood attack which in turn is a type of DoS attack. [3] This attack leaves many connections half-open so no valid connections can be established. As the predictions were that some normal were incorrectly defined as a kind of DoS attack this clearly gives the sysadmin wrong alerts.

### 4.15 Experiment 13

The last few experiments have focused on reducing the training data. What happens if one increases the training data instead? A hypothesis here is that the algorithm’s accuracy is increased, due to more data to train on. In this experiment the training data is increased to 80% of the original data set.

Until now the results have been listed with both the table function in R and the crosstable function.

<table>
<thead>
<tr>
<th>data . testLabels</th>
<th>neptune .</th>
<th>normal .</th>
<th>portsweep .</th>
<th>smurf .</th>
<th>Row Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>back</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>0.010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>0.043</td>
<td>0.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>0.010</td>
<td>0.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>neptune</td>
<td>47</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>0.226</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.226</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>normal</td>
<td>0</td>
<td>43</td>
<td>0</td>
<td>0</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>43</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.15.1 Short analysis

The values for precision and recall for this experiment are:
Smurf – Precision: 100%, Recall: 114/115 = 99.1%.
Normal - Precision: 43/46 = 93.5%, Recall: 100%.
Neptune - Precision: 100%, Recall: 100%.

The prediction here was a bit too much *normal* traffic than actually was the case. One *SMURF* attack was falsely predicted as *normal* traffic. Also one back attack was also falsely predicted as *normal* traffic. False negatives are something one does not want when dealing with *IDS*.

4.16 Experiment 14

The previous experiments focused on reducing and increasing the training data to different levels of the original dataset. Now what happens if one increases the value of\( k \)? The optimal value of \( k \) is discussed in the next
chapter, but this experiment may give a hint. It will be performed with 2/3 of the dataset as the training data as was the case in the first experiment. Increasing \( k \) just a number or two should not have any impact, so in this experiment the \( k \) will be 13. There is a possibility that \( R \) might crash, due to the algorithm becoming more expensive when increasing \( k \).

Then the results.

```r
> CrossTable(x = data.testLabels, y = data_pred, prop.chisq=FALSE)
```

<table>
<thead>
<tr>
<th>Cell Contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N</td>
</tr>
<tr>
<td>N / Row Total</td>
<td></td>
</tr>
<tr>
<td>N / Col Total</td>
<td></td>
</tr>
<tr>
<td>N / Table Total</td>
<td></td>
</tr>
</tbody>
</table>

Total Observations in Table: 326

<table>
<thead>
<tr>
<th>data.testLabels</th>
<th>data_pred</th>
<th>normal</th>
<th>smurf</th>
<th>Row Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>back.</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.006</td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>0.000</td>
<td>0.011</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>0.000</td>
<td>0.006</td>
<td></td>
</tr>
<tr>
<td>neptune.</td>
<td>72</td>
<td>0</td>
<td>0</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.221</td>
</tr>
<tr>
<td></td>
<td>0.878</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.221</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>normal.</td>
<td>7</td>
<td>61</td>
<td>0</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>0.103</td>
<td>0.897</td>
<td>0.000</td>
<td>0.209</td>
</tr>
<tr>
<td></td>
<td>0.085</td>
<td>1.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.021</td>
<td>0.187</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>portsweep.</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.009</td>
</tr>
<tr>
<td></td>
<td>0.037</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.009</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>smurf.</td>
<td>0</td>
<td>0</td>
<td>181</td>
<td>181</td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.555</td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>0.000</td>
<td>0.989</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>0.000</td>
<td>0.555</td>
<td></td>
</tr>
<tr>
<td>Column Total</td>
<td>82</td>
<td>61</td>
<td>183</td>
<td>326</td>
</tr>
<tr>
<td></td>
<td>0.252</td>
<td>0.187</td>
<td>0.561</td>
<td></td>
</tr>
</tbody>
</table>

42
4.16.1 Short analysis

The values for precision and recall for this experiment are:
Smurf - Precision: 181/183 = 98.9%, Recall: 100%.
Normal - Precision: 100%, Recall: 61/68 = 89.7%.
Neptune: - Precision: 72/82 = 87.8%, Recall: 100%.

As one can see the most significant errors in this prediction are too many neptune attacks and too few normal traffic.
Chapter 5

Discussion

In this chapter a more in depth discussion of the experiments in chapter 4 will be done. Also some mathematical predictions of how the IRIS dataset can be used to make a prediction of how well the algorithms tested on the IRIS dataset will perform on the KDD dataset.

5.1 KNN algorithm

So what does the KNN algorithm really do? Mathematically implemented in R the algorithm uses Euclidan distance to determine the nearest neighbour.

\[
d(p,q) = \sqrt{(q1-p1)^2 + (q2-p2)^2}
\]  

(5.1)

The formula above describes the Euclidan distance between two points in a two dimensional plane. The equation is based on the Pytaghorian theorem. The equation above be rewritten into:

\[
d(p,q) = \sqrt{\sum_{i=1}^{k}(x_i-y_i)^2}
\]  

(5.2)

The value of \(k\) is of particular importance. The less the value of \(k\) the less neighbours are used. It is therefore very important to find the optimal value of \(k\) in the algorithm. So what is the optimal value of \(k\) in the KNN algorithm? The larger the value of \(k\) makes the algorithm costly to run. With a smaller value there will be more noise in the results. Running KNN with a \(k\) value of 100 made R studio crash. The predictions would have been quite accurate though, but at a huge cost. A rule of thumb seems to be \(k = \sqrt{n}\) is the best value of \(k\). \(N\) is here the number of features in the dataset. A feature in Machine Learning is a property or a value that is being observed. An example of a feature in the datasets used are the species in the IRIS dataset and attack type in the KDD dataset. In the IRIS dataset there were three species. \(\sqrt{3} = 1.73\) so a value of either 1 or 2 should be optimal. However, the value of 2 is an even value and 1 may not give the optimal result. So the value in most of the experiments involving the IRIS dataset was 3. The number 3 is small enough to predict nearly correct result, but it
is not high enough to make the cost of the algorithm so high that running the algorithm takes a lot of resources in terms of hardware. The number of features in the modified KDD dataset is 2. \( \sqrt{2} = 1 \), so according to the formula 1 would have been the optimal value of \( k \) in this case. However, again due to the reasons mentioned above the value of \( k \) in most of the experiments were 3. In the full KDD dataset there are 23 features so the optimal value of \( k \) would have been \( \sqrt{23} = 4.79 \) or rounded up to 5 to avoid ties.

5.2 Discussions of the experiments

Many experiments have been done with more or less three different datasets. The first experiments were with the built in IRIS dataset and the other ones were with a modified KDD-cup 99 and a sample of 1000 lines from the full KDD-cup 99 dataset. To know the exact accuracy and performance it is important to calculate the error rate in each scenario. The KNN algorithm does not make any assumptions of the data beforehand, hence making the algorithm quite powerful in terms of predictions.

5.2.1 Experiment 1

A quick recap of the results from experiment 1

```
> table (iris_knn)
iris_knn
  setosa versicolor virginica
     11     20     17

> table (iris.testlabel)
iris.testlabel
  setosa versicolor virginica
     11     21     16
```

The prediction was correct in predicting setosa, but not the two others. The total number of observations was 48. The prediction was 17 on the virginica species, but the actual number was 16. Likewise the prediction was 20 of versicolor, but the actual number was 21. This makes the prediction wrong in 2/48 of the observations in this experiment or in 4,2% of the cases. In other terms the algorithm predicted correctly in about 96% of the cases. In this case 2/3 of the dataset was used as a training data and it was also a very small dataset. The question here if this margin of error scales to a larger dataset. This requires to take a closer look at the similar experiment with the 1000 samples from the KDD dataset.

5.2.2 Experiment 10

So what happens when the dataset is bigger? Experiment 1 proved that the predictions were right in about 96% of the cases. If this scales to bigger
datasets, the KNN algorithm is a viable option when choosing a machine-learning algorithm to work with intrusion-detection-systems. As the table of experiment 10 is too large to fit into for formatting reasons, the results will be just be written out in plain text. The prediction was 180 SMURF attacks, 75 Neptune attacks, 71 normal traffic and no other attacks. This is a total of 326 observations. The correct observations were 181 SMURF attacks, 68 normal traffic, 72 neptune attacks, 2 loadmodule attacks and 3 portsweep attacks. This means there was an error in 7 of the 326 packets in the prediction. In other terms this means that the algorithm predicted only 2,14% wrong or around 98% correctly. If this is the case in the other experiments as well using the KNN algorithm in regards to IDS will be quite powerful.

5.2.3 Experiment 2

In this experiment the training data of the IRIS dataset was reduced from 2/3 of the original dataset to 50%. The results were as following.

```
> table(knn50)
knn50
   setosa versicolor virginica
     25       27       28
> table(iris.testlabel50)
iris.testlabel50
   setosa versicolor virginica
     25       30       25
```

The total number of observations in total was 80. The number of wrong predictions was 6. The algorithm predicted incorrectly in 7,5% of the cases. This is probably due to the reduced amount of training data. Does the margin of error scale to a larger dataset? A closer look at experiment 11 should tell that.

5.2.4 Experiment 11

Experiment 11 narrowed the training data of the KDD dataset down to 50% of the original dataset. The number of predicted SMURF attacks were 279, the number of predicted neptune attacks were 101, the number of predicted normal were 94, the number of predicted satan attacks were 1. In total the number of observations were 475. The facts here were the number of SMURF attacks were 279, the number of neptune attacks were 99, the number of normal traffic were 92, the number of portsweep attacks were 2, the number of satan attacks were 1 and the number of ipsweep were 2. This means that there were 4 wrong predictions in total or 0,84%. This is interesting. In experiment 2 the error rate was 7,5%. What is the explanation of this? With reduced amount of training data the logical hypothesis should be that the error rate should be larger not smaller. To be certain that this is the case, the experiment should be rerun. The full source
code is shown, just to document every step in case it was some errors in experiment 11.

```r
> data_normalized = data
> set.seed(1234)
> ind <- sample(2, nrow(data_normalized), replace=TRUE, prob=c(0.50, 0.50))
> data.training <- data_normalized[ind==1, 1:41]
> data.test <- data_normalized[ind==2, 1:41]
> data.trainLabels <- data[ind==1, 42]
> data.testLabels <- data[ind==2, 42]
> data_pred <- knn(train = data.training, test = data.test, cl =
  data.trainLabels, k=3)
```

The predictions:

```r
> table(data_pred)
```

```
back. buffer_overflow. ftp_write. guess_passwd . imap. land
 ipsweep. loadmodule. 0 0 0 0
 0 0 0 0
multihop. neptune. rmap. normal
  perl. phf. pod.
  portswipe. 0 118 0 0
  95 0 0 0
rootkit. satan. smurf. spy
  teardrop. warezclient. warezmaster.
  0 1 301 0
  0 0 0
```

The facts:

```r
> table(data.testLabels)
```

```
data.testLabels
back. buffer_overflow. ftp_write. guess_passwd . imap. land
  ipsweep. loadmodule.
  2 0 0 0 1
  0 0 0 0
multihop. neptune. rmap. normal
  perl. phf. pod.
  portswipe. 0 116 0 0
  91 0 0 0
rootkit. satan. smurf. spy
  teardrop. warezclient. warezmaster.
  0 1 302 0
  0 2 0 0
```
Now as one can see here there were 7 wrong predictions out of 515 observations making the error rate 1.3%. Now this is interesting. Why was the error rate much less here than in the similar experiment using the IRIS dataset? A theory is that the features of the IRIS dataset is evenly distributed (50 of each species), whereas in the KDD has much more of the SMURF attack than anything else. One can in a way say that the features in the KDD dataset is somehow skewed. The SMURF attack is represented in 2,807,886 observations in a dataset of 4.8 million. Discovering most of the SMURF attacks will be much easier. However, it is more likely that since the IRIS dataset is much smaller, three or four errors will of course have much more impact on the results than in a larger dataset. If there are 10 observations and 2 of them are wrong, then the error is 20%. A further discussion of some experiments are needed to draw this conclusion, but if this small error margin proves to be the case throughout, there is no doubt that the KNN algorithm is powerful on even bigger datasets.

5.2.5 Experiment 7, 8 and 9

These experiments were just done with two features of the KDD dataset, so the error margin is very small. Here are the predictions and the results of experiment 7, 8 and 9.

```
> table(data_pred)

 data_pred
buffer_overflow. normal. 0 3279

> table(data.testLabels)

     data.testLabels
buffer_overflow. normal. 1 3278

> table(data_pred)

     data_pred
buffer_overflow. normal. 0 4990

> table(data.testLabels)

     data.testLabels
buffer_overflow. normal. 1 4989

> table(data_pred)

     data_pred
buffer_overflow. normal. 0 8014

> table(data.testLabels)

     data.testLabels
buffer_overflow. normal. 2 8012
```

The margin of error here is less than 1% in all cases, but as the sample of that dataset contains only two of the features these three experiments can
be disregarded. The other reason for disregarding these three experiments is that one feature is very overrepresented compared to the other feature in the whole dataset. The margin of error then becomes very small. It is, however, important to mention from these three experiments that skewing the data may not give any idea how well the predictions are.

5.2.6 Experiment 3

In this experiment the training data in the IRIS dataset was further reduced to 20%. The results are below.

```r
> table (knn_20)
knn_20
  setosa versicolor virginica
  38    40    47
> table (iris.testLabel20)
iris.testLabel20
  setosa versicolor virginica
  38    42    45
```

The total number of observations here was 125 and with 4 wrong predictions gives a margin of error of 3.2%. Now this is interesting. In the experiment 2 the margin of error was 7.5%. This does not need to mean anything special. In theory the margin of error should have been larger with less training data to train the algorithm on. Does this margin of error scale with the KDD dataset? Now in the other experiments the margin of error were less than in the IRIS dataset. This is due to the number of entries in the dataset which is 150 in the IRIS set and a sample of 1000 in the KDD dataset. A further look at the similar experiment with the KDD dataset is needed. If the margin of error is less than in the experiment with 50%, the hypothesis that the accuracy of the algorithm becomes less with less training data to work on might be invalid.

5.2.7 Experiment 12

In the IRIS the margin of error was less with 20% of the dataset as training data than with 50% of the training data. It is interesting to see if this is the case with the KDD dataset as well. If this is the case these experiments might have to be done again. The predictions in experiment 12 were 456 SMURF attacks, 198 NEPTUNE attacks and 155 normal traffic. The facts were 456 SMURF attacks, 182 Neptune attacks, 163 normal traffic, 3 SATAN attacks, 2 loadmodule attacks, 2 portsweep attacks and 1 ipsweep attack. In total there were 809 observations. In total the number of wrong predictions here are 24. That gives a margin of error of 2.97%. Compared to experiment 11 which had a margin of error of under 1%, this means that the hypothesis is more or less correct in this case. In the experiment with the IRIS dataset the error margin was much less with 20% training data than with the 50%. The reason for that might just be a "noisy" experiment, but there may also
be other reasons for that. Now it is time to take a closer look at the value of $K$.

### 5.2.8 Experiment 4, 5 and 6

These are the tree experiments which focused on the value of $k$ when using the IRIS dataset. The results of these are listed below. Experiment 4:

```r
> knn_k <- knn(train=iris_train_20, test=iris_test_20, cl=iris_train_label_20, k=5)
> table(knn_k)

knn_k  
setosa versicolor virginica
 38   40   47
```

Experiment 5:

```r
> knn_k_2_3 <- knn(train=iris_training, test=iris_test, cl=iris_train_label, k=5)
> table(knn_k_2_3)

knn_k_2_3  
setosa versicolor virginica
 11   21   16
```

Experiment 6:

```r
> knn_k_20 <- knn(train=iris_training, test=iris_test, cl=iris_train_label, k=20)
> table(knn_k_20)

knn_k_20  
setosa versicolor virginica
 11   24   13
```

In experiment 4 there were 4 errors, in experiment 5 there predictions were 100% correct while in experiment 6 there were 6 errors. This means that in experiment 4 the error margin was 3.2% and in experiment 6 was 12.5% which is the highest in the experiment encountered. In experiment 4, the training data used was 20% of the original dataset, in experiment 5 the
training data was 2/3 of the original dataset, whereas in experiment 6, the training data was back to 20% of the original dataset. With 2/3 of the original dataset used for training and the value of \( k = 5 \), the predictions were 100% accurate. It is a quite interesting observation that with a value of \( k = 20 \) the predictions were the most inaccurate in all of the experiments. The predictions should in theory at least become more accurate when increasing the value of \( k \). This may have something to do with the way the KNN algorithm determines the nearest neighbour. The KNN determines a classification by a majority vote. With a \( k \) value of 20, there are many neighbours to classify and this may give noise in the predictions. As mentioned earlier in this chapter it is very important to find the optimal value of \( k \) to make the algorithm both cost efficient and to make as good predictions as possible.

5.2.9 Experiment 14

In this experiment \( k \) was increased to 13 on the KDD dataset, but with 2/3 used as training data. The predictions here ended up with 82 Neptune attacks, 61 normal traffic and 183 SMURF attacks, in total 326 observations. However, the facts were 72 Neptune attacks, 68 Normal traffic, 181 SMURF attacks, 2 back attacks and 3 portsweep attacks. This gives us a margin of error 12/326 or around 3.68%. This is close to the margin of error encountered in the other experiments with the KDD dataset.

5.2.10 General observations

The KNN algorithm applied on the KDD dataset generally had a low number of errors whereas on the IRIS dataset the margin of error greatly varied. This is of course due to the number of entries in each dataset. The IRIS dataset had 150 observations whereas the two modified datasets from KDD had 10000 and 1000 observations. The reason for why only 1000 entries from the entire KDD dataset was that having too many entries resulted the algorithm yielding too many ties when predicting. Without drawing any final conclusion on this, this may make the KNN algorithm unusable on very large datasets. Still there are many examples out there that researches have used machine-learning algorithms on this particular dataset without any problem, so the problems encountered here may be solved if the dataset was tuned correctly to fit into the R function.

5.2.11 Other algorithms

There are other algorithms which could have been used on the datasets. The KNN algorithm is quite simple. What other algorithms could have been used on the three datasets used? All data in the three datasets are labelled meaning that the datasets have certain properties to apply the algorithm on. The case in the IRIS dataset was the flowers and the case in the KDD dataset was the attack types. If the data was unlabeled, the KNN algorithm would be unsuitable. In this case a unsupervised
learning algorithm would be more suitable instead. The main difference between supervised and unsupervised learning is that in supervised learning the input labels are known, whereas in unsupervised learning they are not. In these cases other supervised learning algorithms are the most suitable ones. However, the KDD dataset can be downloaded as a completely unlabeled dataset, so it is possible to use unsupervised algorithms as well. R has support for the K-means algorithm for instance which is a unsupervised learning algorithm. However, since the details are discussed in chapter 3 there is not much need to go into too much detail here. It is difficult to tell how good a prediction from one of the algorithms will be without actually performing experiments with them. Sabhanhi and Serpen [13] did apply more algorithms to the KDD dataset with both supervised and unsupervised learning. As shown in the paper there is not much difference between the types of algorithms when it comes to the margin of error.

5.2.12 Training data

So far in this thesis the phrase training data has been mentioned many times, but has not been discussed properly. It is important to understand what the training data means when using the KNN algorithm and what impact the training data has on the results. The training data in Machine Learning is used to train the algorithm and to build a model. In the experiments done, the amount of training data was 66%, 50% and 20%. It is very important that the training data used contains all features in the dataset, otherwise the algorithm will predict something completely else. The R code for choosing the training and test samples does this and is even resembled in experiments 7, 8 and 9 where the sample was completely different than in the original dataset.
Chapter 6

Conclusion and future work

In this thesis, we investigated using the KNN algorithm on three different datasets with emphasis on discovering attacks in a dataset extracted from Tcpdump. The aim of this thesis was defined as "How does the value of K and different training data affect the performance of the KNN algorithm in different training sets?". A total of 14 experiments were performed. The experiments involved varying the following factors: size of data set, relative size of the training data set and the value of the neighbor of neighbors k. We have seen that there was little variation between the results of the experiments and this indicates that the KNN algorithm is quite robust with respect to the tuning parameters and the size of the data set for predicting observations in a data set. In this thesis three different datasets have been used: IRIS and two versions of the KDD cup dataset. The KNN algorithm proved to be quite powerful on both small and larger datasets with a low average margin of error. Even with reducing the training data to 20%, the margin of error proved to stay quite low. Also, the value of k did not have significant impact other than in experiment 6, which had a margin of error of over 12%. Based on the results in this thesis, one can overall quite safely assume that the KNN algorithm can be applied to detect attacks without much of a concern. As mentioned in the introduction, Machine Learning is not widely used in IDS today, but as results have shown, the algorithms used in both research papers in the area and in this thesis proves its potential. In this thesis we have seen examples of false negatives and we know that even one false negative can cause serious problems to IT infrastructure. Therefore, it seems unlikely that Machine Learning will replace traditional IDS implementation in the near future. Nevertheless, there is no doubt that Machine Learning can enhance current IDS by using it in conjunction with classical rule-based IDS. Further research is therefore necessary to ensure that false negatives can be prevented.

6.1 Future work

This thesis resorted to KNN in three different datasets with the primary focus on the KDD dataset. As a future work, more sophisticated algorithms can be examined. Furthermore, it would be interesting to integrate KNN
within an existing IDS, for instance SNORT, so that to test its efficiency in real-life traffic. Even though Machine Learning algorithms are widely used in nearly similar scenarios, for instance email SPAM filtering, we believe there is still a long way to go before seeing commercial Machine Learning based IDS.
Appendix

Experiment 1

```r
set.seed(1234)
iris_sample <- sample(2, nrow(iris), replace=TRUE, prob=c(0.67, 0.33))
```

Experiment 2

```r
set.seed(1234)
iris_sample_50 <- sample(2, nrow(iris), replace=TRUE, prob=c(0.50, 0.50))
iris.train50 <- iris[iris_sample_50==1, 1:4]
iris.test50 <- iris[iris_sample_50==2, 1:4]
iris.trainlabel50 <- iris[iris_sample_50==1,5]
iris.testlabel50 <- iris[iris_sample_50==2,5]

knn50 <- knn(train=iris.train50, test=iris.test50, cl=iris.trainlabel50, k=3)
```

Experiment 3

```r
set.seed(1234)
iris_20 <- sample(2, nrow(iris), replace=TRUE, prob=c(0.20, 0.80))
iris_train_20 <- iris[iris_20==1, 1:4]
iris_test_20 <- iris[iris_20==2, 1:4]
iris.trainLabel20 <- iris[iris_20==1, 5]
iris.testLabel20 <- iris[iris_20==2, 5]

knn_20 <- knn(train=iris_train_20, test=iris_test_20, cl=iris.trainLabel20, k=3)
```

Experiment 4
Experiment 5

```r
knn_k_2<>knn(train=iris_train_20, test=iris_test_20, cl=iris_trainLabel20, k=5)
```

Experiment 6

```r
knn_k_2_3<-knn(train=iris.training, test=iris.test, cl=iris.trainLabel, k=5)
> table(knn_k_2_3)
  setosa versicolor virginica
   11    21     16
> table(iris.testlabel)
  setosa versicolor virginica
   11    21     16
```

Experiment 7

```r
ind <- sample(2, nrow(kdd_sample), replace=TRUE, prob=c(0.67, 0.33))
kdd.training <- kdd_sample[ind==1, 1:41]
kdd.testing <- kdd_sample[ind==2, 1:41]
kdd.trainlabels <- kdd_sample[ind==1, 42]
kdd.testlabels <- kdd_sample[ind==2, 42]
kdd_knn <- knn(train = kdd.training, test = kdd.testing, cl = kdd.trainlabels, k=3)
Error in knn(train = kdd.training, test = kdd.testing, cl = kdd.trainlabels, : too many ties in knn
```
> data = small1
> data_normalized=data
> set.seed(1234)
> ind <- sample(2, nrow(data_normalized), replace=TRUE, prob=c(0.67, 0.33))
> data.training <- data_normalized[ind==1, 1:41]
> data.test <- data_normalized[ind==2, 1:41]
> data.trainLabels <- data[ind==1, 42]
> data.testLabels <- data[ind==2, 42]
> data.pred <- knn(train = data.training, test = data.test, cl = data.trainLabels, k=3)

Experiment 8

> set.seed(1234)
> ind <- sample(2, nrow(data_normalized), replace=TRUE, prob=c(0.50, 0.50))
> data.training <- data_normalized[ind==1, 1:41]
> data.test <- data_normalized[ind==2, 1:41]
> data.trainLabels <- data[ind==1, 42]
> data.testLabels <- data[ind==2, 42]
> data.pred <- knn(train = data.training, test = data.test, cl = data.trainLabels, k=3)

Experiment 9

> ind <- sample(2, nrow(data_normalized), replace=TRUE, prob=c(0.20, 0.80))
> data.training <- data_normalized[ind==1, 1:41]
> data.test <- data_normalized[ind==2, 1:41]
> data.trainLabels <- data[ind==1, 42]
> data.testLabels <- data[ind==2, 42]
> data.pred <- knn(train = data.training, test = data.test, cl = data.trainLabels, k=3)

Experiment 10

> data = kddcup.data[sample(nrow(kddcup.data), 1000),]
> set.seed(1234)
> data_normalized=data
> ind <- sample(2, nrow(data_normalized), replace=TRUE, prob=c(0.67, 0.33))
Experiment 11

```r
> ind <- sample(2, nrow(data_normalized), replace=TRUE, prob=c(0.50, 0.50))
> data.training <- data_normalized[ind==1, 1:41]
> set.seed(1234)
> data.training <- data_normalized[ind==1, 1:41]
> data.test <- data_normalized[ind==2, 1:41]
> data.trainLabels <- data[ind==1, 42]
> data.testLabels <- data[ind==2, 42]
> data_pred <- knn(train = data.trainings, test = data.test, cl = data.trainLabels, k=3)
```

Experiment 12

```r
> ind <- sample(2, nrow(data_normalized), replace=TRUE, prob=c(0.20, 0.80))
> data.training <- data_normalized[ind==1, 1:41]
> data.test <- data_normalized[ind==2, 1:41]
> data.trainLabels <- data[ind==1, 42]
> data.testLabels <- data[ind==2, 42]
> data_pred <- knn(train = data.trainings, test = data.test, cl = data.trainLabels, k=3)
```

Experiment 13

```r
> set.seed(1234)
> ind <- sample(2, nrow(data_normalized), replace=TRUE, prob=c(0.80, 0.20))
> data.training <- data_normalized[ind==1, 1:41]
> data.test <- data_normalized[ind==2, 1:41]
> data.trainLabels <- data[ind==1, 42]
> data.testLabels <- data[ind==2, 42]
```
Experiment 14

```r
> set.seed(1234)
> ind <- sample(2, nrow(data_normalized), replace=TRUE, prob=c(0.67, 0.33))
> data.training <- data_normalized[ind==1, 1:41]
> data.test <- data_normalized[ind==2, 1:41]
> data.trainLabels <- data[ind==1, 42]
> data.testLabels <- data[ind==2, 42]
> data_pred <- knn(train = data.training, test = data.test, cl = data.trainLabels, k=13)
```
Bibliography


