Universiteit Leiden
Opleiding Informatica

Anomaly Detection

in Cybersecurity

Name:	Simone van Veen
Date:	02/07/2016
1st supervisor:	Dr. W.J. Kowalczyk
2nd supervisor:	Dr. Amr Ali-Eldin

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Anomaly Detection in Cybersecurity

Simone van Veen

Abstract

The increase in use of computers and computer networks has made it very important to develop reliable intrusion detection systems (IDS). These systems are used to detect attempts to gain access or cause damage to a computer network. An IDS is confronted with large amounts of data. The majority of intrusion detection systems use machine learning techniques to find indications of suspicious behaviour in this data.

Supervised learning is a commonly used machine learning approach for network intrusion detection. The objective of supervised learning is to train a classifier on a labelled dataset and use it to predict in which class an observation belongs. In the case of an IDS, for example, a classifier can be used to predict whether an observation represents an attack or not.

Supervised learning techniques can achieve high accuracy when used to detect attacks. However, the need for a labelled dataset is a big disadvantage. Real data is sensitive and therefore rarely made public and creating a dataset takes much time. Labelling data is expensive and only done in exceptional cases. Even if a labelled dataset is created its usefulness will rapidly decrease due to the constant development of new threats.

Unsupervised learning is an approach that does not use labelled datasets. This thesis will focus on unsupervised anomaly detection. The goal of anomaly detection is to identify which observations in a dataset deviate from normal observations. These deviating instances are called outliers or anomalies. The aim of this thesis is to examine to which extent unsupervised anomaly detection can be used to detect attacks. This is done by applying statistical methods and a distance based algorithm to the NSL-KDD dataset. The results show that these approaches are not suited to detect attacks in this dataset because there is no clear distinction between normal and malicious instances.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the tremendous increase in use of computers and computer networks in the last decade it has become more and more important to protect computer systems. Not only large companies and banks are under constant attack from hackers; hospitals, healthcare providers and schools are also targets. Cybersecurity protects our computer systems from a number of threats such as unwanted network access or data theft.

Network security is one of the main areas of focus in cybersecurity. To protect a network from unwanted access or attack it is crucial to detect any attempt to gain access or cause damage. An intrusion detection system (IDS) monitors network traffic or system activities and searches for such malicious behaviour. All data is kept in log files. Logging is important because it helps to provide information about suspicious behaviour. An IDS is confronted with large amounts of data. The majority of intrusion detection systems use machine learning techniques to find indications of suspicious behaviour in this data.

Supervised learning is a commonly used machine learning approach for network intrusion detection. The objective of supervised learning is to train a classifier on a labelled dataset and use it to predict in which class an observation belongs. In the case of an IDS, for example, a classifier can be used to predict whether an observation represents an attack or not.

The main disadvantage of this approach is the lack of labelled datasets. Real data is sensitive and therefore rarely made public and creating a dataset takes much time. Labelling data is expensive and only done in exceptional cases. Even if a good dataset is created its usefulness will rapidly decrease due to the constant development of new threats.

Unsupervised learning is an approach that does not use labelled datasets. It aims at modelling common, general properties of data. This thesis will focus on unsupervised anomaly detection. The goal of anomaly detection is to identify which observations in a dataset deviate from normal observations. These deviating instances are called outliers or anomalies.

Anomalies can be divided into three categories. The first category consists of point anomalies which are

individual instances that are outliers with respect to other data. The second category contains instances that are only outliers in a specific context. These are called contextual anomalies. Collective anomalies make up the last category. Anomalies of this type are groups of instances that are anomalous with respect to the rest of the dataset. Individual instances in these groups are not necessarily outliers themselves. [CBK07]

If anomaly detection can be used to detect attacks on a network the problem of labelled datasets would be solved. Anomaly detection would also be able to detect unknown attacks if they differ enough from normal instances. The aim of this thesis is to examine to which extent anomaly detection can be used to detect attacks. This is done by applying statistical methods and a distance based algorithm to the NSL-KDD dataset [0e15].

The rest of this thesis is organized as follows. Chapter 2 discusses related work. Chapter 3 provides a description of the data that is used. Chapter 4 explains the methods used. In chapter 5 the results are evaluated and chapter 6 contains some conclusions.

Chapter 2

Related Work

As it becomes increasingly important to develop reliable intrusion detection systems there has been done a lot of research on this subject [CBK07]. To facilitate this research several datasets have been created. Creating a dataset can be done in various ways. One approach is to simulate normal network traffic and attacks. Another way would be to set up a honeypot. A honeypot is a computer system that is set up to attract hackers that try to attack or gain access to the system. The network traffic from a honeypot often contains a lot of attacks. It is also possible to use a dataset that consists of real-life data, for example, the data from a company or a bank. The labelling of this data has to be done by a human expert that analyses traffic data and manually labels the relevant records. This causes a problem because the size of these datasets is very large. The sets created using honeypots or real-life networks generally contain gigabytes of data.

The KDDcup99 dataset [IoC99] has been the most widely used set since 1999. The NSL-KDD dataset is an updated version of this dataset. These and other existing datasets are discussed by Bhuyan, Bhattacharyya and Kalita [BBK15].

Several intrusion detection systems have been developed and they all use a certain approach to detect attacks. These approaches are based on supervised or unsupervised learning and can be divided into three categories. The first category is signature detection. Signature detection uses knowledge of known attacks and vulnerabilities to look for patterns that are known to cause security problems. Anomaly detection is the second category. The last category consists of classification-based detection systems. These systems use machine learning techniques to build models on labelled datasets that can be used to classify observations. An overview of these categories and which systems use them is given by Lin [Lin13].

Supervised learning is an often used approach and can achieve high accuracy when used to detect attacks. This is shown by Revathi and Malathi [RM13] and Dhanabal and Shantharajah [DS15] who provided an overview of the performance of supervised learning techniques on the NSL-KDD dataset. Even though the results are good, the need for a labelled dataset is a big disadvantage of this approach.

Unsupervised learning does not require labelled data. Unsupervised anomaly detection is used to find patterns that deviate from expected behaviour. There are many different anomaly detection techniques. A comprehensive overview is given by Chandola, Banerjee and Kumar [CBK07].

If anomaly detection can be used as a reliable way of detecting network intrusions, the problematic need of labelled datasets would be solved. To evaluate whether a method is reliable multiple criteria can be considered. One of these criteria is the effectiveness. The effectiveness is defined by the true positive rate and the false positive rate. The true positive rate measures the percentage of attacks that are correctly identified as such. The false positive rate is the percentage of normal instances that are incorrectly classified as attack. A reliable intrusion detection system should have a high true positive rate; the percentage of attacks that are detected has to be as high as possible. It is equally important to keep the false positive rate very low. The system have to be very sensitive to keep the amount of false alarms as small as possible. This is a challenging problem as usually attacks are very rare and a system that misclassifies only a very small portion of normal records generates many false alarms. This phenomenon is explained in more detail by Axelsson [Axe].

Portnoy, Eskin and Stolfo [PES] propose a cluster-based unsupervised anomaly detection method and evaluate it using the KDDcup99 dataset. They reach an average detection rate between 40% and 55% with a $1.3\% - 2.3\%$ false positive rate.

Eskin et al. [EAP$^+$] present a geometric framework for unsupervised anomaly detection. They test three different algoritms: a cluster-based estimation, a k-nearest neighbour algorithm and a one class SVM. These algorithms are applied to the KDDcup99 and DARPA [SoMLL] datasets. The data is mapped to a feature space. One of the three algorithms within the framework determines which points are outliers based on the position in that feature space. The results were good for both types of data and for all three algorithms. All three algorithms were able to reach a perfect result on data from the DARPA set. For data from KDDcup99 there was a rather high false positive rate.

This thesis will examine unsupervised anomaly detection by applying statistical methods and a distance based algorithm on the NSL-KDD dataset.

Chapter 3

Description of the Dataset

The dataset used for the experiments in this thesis is the NSL-KDD dataset. This is an improved version of the KDDcup99 dataset which had several problems that affect the performance. One could object that the NSL-KDD dataset is outdated and should therefore not be used but this dataset is used for the following reasons. The data consists of point based instances. There are no time stamps or other attributes that are mainly context specific. Even though the attack types might be old and not often seen these days, the focus of this project is to evaluate the usefulness of anomaly detection-based systems for detecting cyber attacks. The aim is not to develop a model that is based on the attack types but to develop a model that is based on the deviation from normal observations.
The dataset consists of a train and a test set. The training set contains 125973 records, the test set 22544. Neither contain duplicate records. The data records are vectors of 41 attributes that represent network connection instances. All instances are labelled as normal or as a specific attack type.

3.1 Attributes

The attributes in the dataset are of mixed types. An overview of the attributes is given in table 3.1. This and other information about the attributes can be found in [DS15].

Type	Attributes	Description
Binary	Land	1 if source and destination IP addresses and port numbers are equal, 0 otherwise
	Logged_in	1 if successfully logged in, 0 otherwise
	Root_shell	1 if root access, 0 otherwise
	Is_host_login	1 if host (root, admin) login, 0 otherwise
	Is_guest_login	1 if guest login, 0 otherwise
	Su_attempted	1 if su root command attempted, 0 otherwise
Nominal	Protocol_type	Protocol type used in connection
	Service	Service of destination network
	Flag	Connection status
Numeric	Duration	Duration of the connection
	Src_bytes	Number of bytes transferred from source to destination
	Dst_bytes	Number of bytes transferred from destination to source
	Wrong_fragment	Number of wrong fragments in connection
	Urgent	Number of packets with urgent bit activated
	Hot	Number of 'hot' actions such as creating and executing a program
	Num_failed_logins	Number of failed login attempts
	Num_compromised	Number of compromised conditions
	Num_root	Number of root accesses or operation performed as root
	Num_file_creations	Number of file creation operations
	Num_shells	Number of shell prompts
	Num_access_files	Number of operations on access control files
	Num_outbound_cmds	Number of outbound commands in ftp session
	Count	Number of connections to the same destination host as the current connection in past two seconds
	Srv_count	Number of connections to the same service (port number) as the current connection in past two seconds
	Serror_rate	Percentage of connections that have activated the flag $s0, s1, s2$ or $s3$ among the connections in count
	Srv_serror_rate	Percentage of connections that have activated the flag $s0, s1, s2$ or $s3$ among the connections in srv_count
	Rerror_rate	Percentage of connections that have activated the flag REJ among the connections in count
	Srv_rerror_rate	Percentage of connections that have activated the flag REJ among the connections in srv_count
	Same_srv_rate	Percentage of connections to the same service among the connections in count
	Diff_srv_rate	Percentage of connections to different services among the connections in count
	Srv_diff_host_rate	Percentage of connections to different destination machines among the connections in srv_count
	Dst_host_count	Number of connections with the same destination host IP address
	Dst_host_srv_count	Number of connections having the same port number
	Dst_host_same_srv_rate	Percentage of connections to the same service among the connections in dst_host_count
	Dst_host_diff_srv_rate	Percentage of connections to different services among the connections in dst_host_count
	Dst_host_same_src_port_rate	Percentage of connections to the same source port among the connections in dst_host_srv_count
	Dst_host_srv_diff_host_rate	Percentage of connections to different destination machines among the connections in dst_host_srv_count
	Dst_host_serror_rate	Percentage of connections that activated the flag $s0, s1, s2$ or $s3$ among connections in dst_host_count
	Dst_host_srv_serror_rate	Percentage of connections that activated the flag $s0, s1, s2$ or $s3$ among connections in dst_host_srv_count
	Dst_host_rerror_rate	Percentage of connections that activated the flag REJ among the connections in dst_host_count
	Dst_host_srv_rerror_rate	Percentage of connections that activated the flag REJ among the connections in dst_host_srv_count

Table 3.1: Attributes in NSL-KDD dataset.

3.2 Attack types

The attacks in the dataset can be divided in the following four categories:

- R2L: Remote to local attacks aim to gain local access to a machine by intruding into a remote machine.
- U2R: User to root attacks are used to gain root privileges by logging in to a normal user account and exploiting vulnerabilities.
- Probing: The goal of probing attacks is to gain information about the victim.
- DoS: Denial of Service attacks exhaust the victim's resources so that handling legitimate requests becomes impossible.

Table 3.2 contains these four categories and all the corresponding attack types from the dataset.

Type	Attack
R2L	Ftp_write, Guess_password, Httptunnel, Imap, Multihop, Named, Phf, Sendmail, Snmpgetattack, Snmpguess, Spy, Warezclient, Warezmaster, Xlock, Xsnoop
U2R	Buffer_overflow, Loadmodule, Perl, Ps, Rootkit, Sqlattack, Xterm
Probing	Ipsweep, Mscan, Nmap, Portsweep, Saint, Satan
DoS	Apache2, Back, Land, Mailbomb, Neptune, Pod, Processtable, Smurf, Teardrop, Udpstrom, Worm

Table 3.2: Attack types in NSL-KDD dataset.

The percentage of attacks in the datasets is unrealistically large. This could cause a problem for some algorithms, especially these that are focussed on anomaly detection. Attacks will only be detected if they are outliers which they are not if there is a large number of them. To deal with this problem, subsets are created that consist of 1% of attacks and of 99% of instances that represent normal, non-malicious behaviour. The distribution between normals and attacks in the training and test set is as follows:

Type	Train	Test
R2L	995	2885
U2R	52	67
Probing	11656	2421
DoS	45927	7460
Normal	67343	9711
Total:	125973	22544

Table 3.3: Instances in training and test set.

The training set and the test set do not contain the same attacks. The training set has two attacks, spy and warezclient, that are not in the test set. Attack types that only appear in the test set are: apache2, httptunnel, mailbomb, mscan, named, processtable, ps, saint, sendmail, snmpgetattack, snmpguess, sqlattack, udpstorm, worm, xlock, xsnoop and xterm. The remaining attack types are in both sets.

Chapter 4

Methods

The aim of this thesis is to examine the usefulness of unsupervised anomaly detection for the detection of cyber attacks. The NSL-KDD dataset that is used for the experiments is a labelled dataset. The target variable, attack type, will however not be used to build models. It will only be used to compare the output of the unsupervised models with the actual labels so that the effectiveness of the models can be evaluated. R is used to explore the data and to implement the necessary algorithms.

4.1 Exploring the dataset

In order to get a better understanding of the dataset the predictive strength of each attribute is investigated. For each attribute in the training set a stacked bar chart is created showing the distribution of attacks and normal, non-malicious records. Besides making plots there are also some statistics calculated. As every attribute contains a few typical values that occur in most records, the records with these typical values are filtered out as there is no hope in detecting outliers there. For all attributes the percentage of attacks and normal instances in the top 5% and 10% of least frequent values is calculated to determine if the distribution in this subset is similar to the distribution in the whole set. The subset is created by removing all values with an added frequency higher than or equal to 95% or 90% respectively.
Aside from this a prediction is made for each numerical attribute in the test set by building a random forest of 50 trees. Numerical attributes in this case are all non-binary attributes with numerical values. The random forest is based on the other 40 attributes using the values from the training set. This way the relationship between attributes becomes more clear. Random forest is used for this because it is very precise and able to handle large amounts of data.
Besides looking at one attribute at the time there are also some calculations done for all pairs of attributes

in the training set. The number of attacks and normal observations are counted for the 10% and 25% least frequent values from both attributes combined if there are at least 1% of all instances left. For example: when combining a binary attribute with value 0 that occurs in 99% of instances and a numerical attribute that also has a value that occurs in 99% of instances, the value 1 of the binary attribute may not occur in the instances that are left after removing the most frequent value from the numeric attribute.

4.2 Extreme Values

The differences between the predictions made by the random forest algorithm and the actual values of the instances are calculated. These differences are used to calculate the z-scores and interquartile range (IQR). The z-score method for detecting outliers is based on the assumption that when an instance is more than 3 standard deviations away from the mean it is most likely an outlier. When using this method it is important that the data has a normal distribution. The distribution should be similar to that in figure 4.1 [Ope].

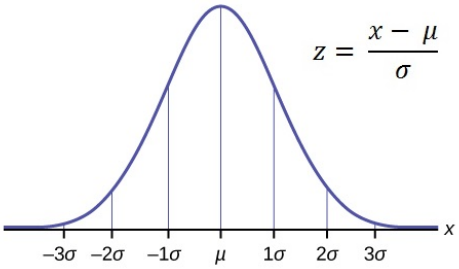

Figure 4.1: Normal distribution.

The z-score is the number of standard deviations from the mean. It is calculated by using the formula:

$$z = \frac{x - \mu}{\sigma} \tag{4.1}$$

where x is the value of an instance, μ is the mean of the population and σ stands for the standard deviation of the population. A negative value indicates a deviation below the mean. When the data is normally distributed 68% of the data has a z-score between -1 and 1. The instances with a z-score below -3 or above 3 cover only 0.3% and are therefore considered anomalies. Using this method, the outliers for each attribute are detected and the true and false positive rates are calculated.

The same approach is taken for the IQR method. This method does not require normally distributed data. To calculate the IQR the data is sorted and then split in four intervals, each containing 25% of the data. The first quartile is called the lower quartile Q_1 and is the point that splits of the lowest 25% of the data. It is the middle number between the lowest record and the median of the data. The second quartile Q_2 is the median and the third or upper quartile Q_3 splits of the top 25%. The IQR is the difference between the third and the

first quartile; $Q_3 - Q_1$. An instance is considered to be anomalous when it has a value more than 1.5 times the IQR below the lower quartile (Q_1) or more than 1.5 times the IQR above the upper quartile (Q_3).

4.3 Distance-based method

The results from the methods discussed in section 4.1 and 4.2 gave the impression that it would be difficult to detect outliers in this dataset. To further investigate this a k-nearest neighbour based algorithm is used. This algorithm can be used to detect outliers using the assumption that normal records are in majority and are close to each other. When mapping all instances to a feature space, the normal records would form relatively big clusters whereas the anomalous instances would lie outside these clusters and could therefore be detected. An example can be seen in figure 4.2 [CBK07]. The clusters $N1$ and $N2$ are groups of similar instances. The points $o1$ and $o2$ as well as the group $O3$ are outliers.

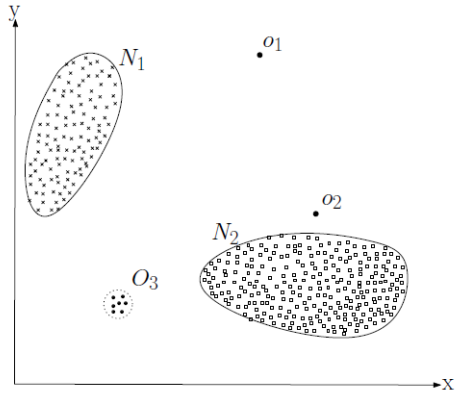

Figure 4.2: Anomalies.

For this approach the distance between all instances has to be calculated. The instances can be viewed as points in R^n. There exist several distance measures but the Euclidean distance is most often used for this. However, the NSL-KDD dataset contains not only numerical but also nominal attributes. Simply computing the Euclidean distance between all values is not possible.

This problem is solved by using the Gower's dissimilarity measure that is implemented in the R package 'cluster' [MRS+16]. The distance d_{ij} between two instances is the weighted mean of the contribution $d(ij,k)$ of each attribute with weight $w_k\delta(ij;k)$. The contribution of numerical attributes is the absolute difference between both values divided by the total range. For nominal and binary attributes the contribution is 0 when the two values are equal and 1 otherwise. w_k is the weight of variable k and can be set if this is desired. The default weight for each attribute is 1. $\delta(ij;k)$ is 0 or 1. It is 0 when the attribute has missing values for one or both instances or when the attribute is asymmetric binary with both values zero. For all other conditions

$\delta(ij;k)$ is 1. This can be summarized by the following formula [SHR]:

$$d_{ij} = d(i,j) = \frac{\sum_{k=1}^{p} w_k \delta(ij;k) d(ij,k)}{\sum_{k=1}^{p} w_k \delta(ij;k)} \tag{4.2}$$

Before the algorithm is applied to the dataset the data has to be preprocessed. All numerical attributes are normalized so that they have a range from 0 to 1. This is done to assure that an attribute with a large range has the same effect on the total distance as an attribute with a smaller range. It is also possible to normalize the data in another way.

After normalizing the data the dissimilarity matrix is computed using the Gower's dissimilarity measure. For each instance the average distance between its k nearest neighbours is computed. This is done for $k = 10$, $k = 100$ and $k = 1000$. The instances with the highest average distances are believed to be anomalous.

This technique is used for the complete test set but also for subsets of this set containing normal instances and instances corresponding to one attack type. When looking at one attack type, the number of attacks is brought back to 1% of the subset when the initial percentage was higher. Aside from this random samples are taken from all attacks to create sets that consists for 1% of attacks and for 99% of normal observations.

Chapter 5

Evaluation

In this chapter the results from the experiments explained in the previous chapter are discussed.

5.1 The predictive strength of each attribute

A stacked bar chart was created for each attribute in the training set to visualize the distribution of attacks and normal instances. Many attributes had a bar chart that looked similar to figure 5.1. Around 99% of instances have the same value so that all instances with other values are detected as outliers.

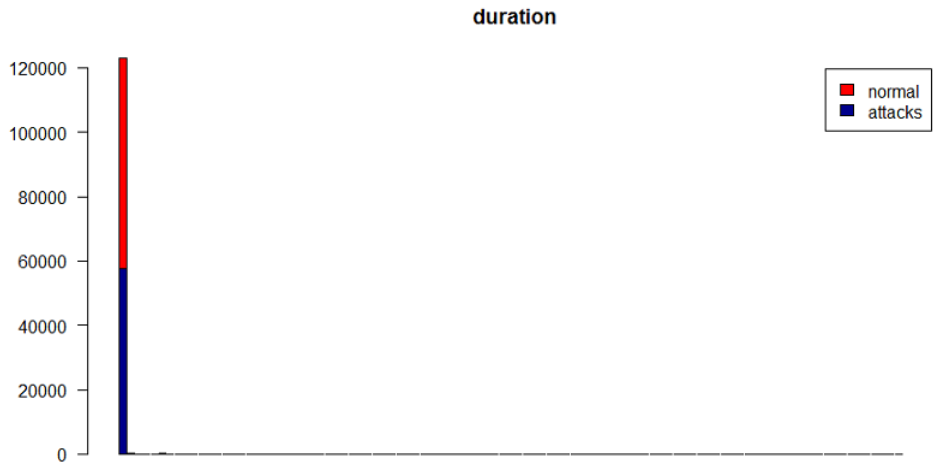

Figure 5.1: Stacked bar chart for duration.

Sometimes these outliers are mostly attacks. An example of this is the attribute same_srv_rate, see figure 5.2. More often the outliers contain more normal instances than attacks, as seen in figure 5.3, or a distribution that is roughly equal.

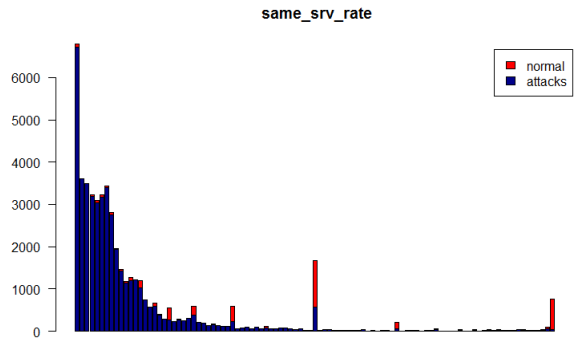

Figure 5.2: Outliers are mostly attacks.

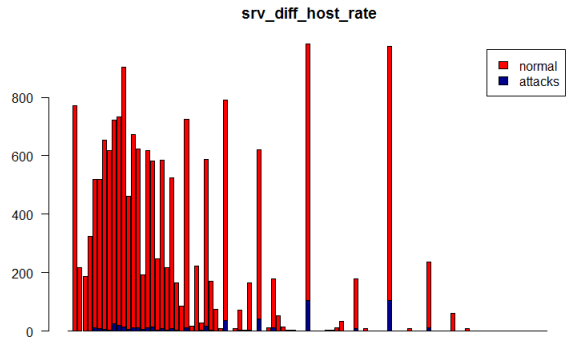

Figure 5.3: Outliers are mostly normal.

For the 5% and 10%, or less when the most frequent values cover more than 95% or 90% of instances respectively, of instances with the least frequent values the percentage of attacks is calculated. The results are summarized in figure 5.4 and 5.5. These figures show that there are many attributes that have less attacks than normal instances with uncommon values. However there are also attributes that seem promising when it comes to detecting attacks.

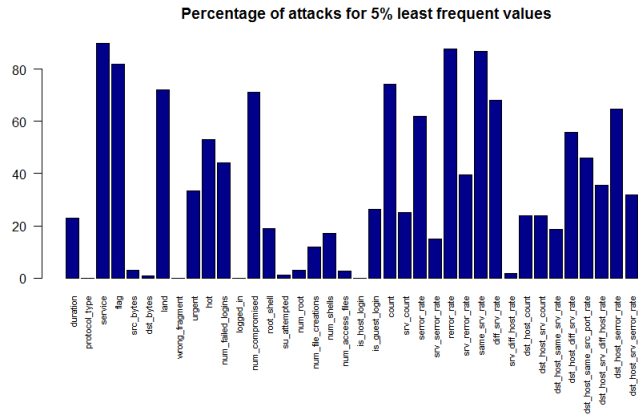

Figure 5.4: 5% least frequent values.

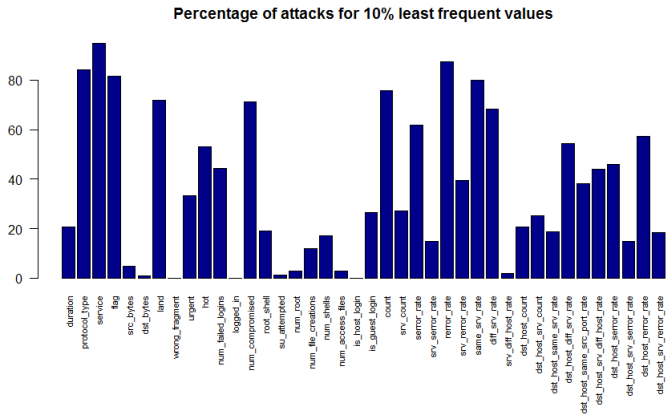

Figure 5.5: 10% least frequent values.

For each numerical attribute the z-score and IQR are calculated based on the difference between the prediction made by a random forest and the actual value of each instance. When using the z-score method to detect outliers it is important that the data is normally distributed. Since the distribution of most attributes looks like figure 5.1, this is not the case for this dataset. This makes it impossible to detect outliers using this method, which can be seen in the results. The true positive rate for each attribute was between 0.00085 and 0.073. The false positive rate ranged from 0.0001 to 0.054. This means that hardly any observations are considered outliers.

Using the IQR method gave slightly better results. The true positive rate ranged from 0.11 to 0.61, the false positive rate form 0.0099 to 0.38. Although the true positive rate went up, the false positive rate is also higher which implies that more observations are believed to be outliers but the detection rate is not necessarily better. In fact the number of false positives is often relatively higher when using the IQR method instead of

the z-score method.

5.2 Combining two attributes

If an instance is detected as an outlier by one attribute it could be an attack. If another attribute also detects this instance as an outlier, the likelihood of it being an attack increases. To test whether this holds for the instances in this dataset the number of attacks and normal observations are counted for the 10% least frequent values in pairs of attributes. Since the amount of attacks in this dataset is high, the same is done for the 25% least frequent values.

For most pairs of attributes there are hardly any outliers. The remaining pairs mostly have a very high false positive rate. However there are a few pairs with a very high true positive rate. Almost all of these pairs involve the attributes service, flag, rerror_rate, same_srv_rate, srv_rerror_rate, diff_srv_rate and count. The attacks detected using this method are often the same for each pair. These attacks are neptune, smurf, portsweep, teardrop, ipsweep, nmap and satan. This group includes all types of probing attacks in the training set and some DoS attacks. Since neptune makes up a large part of all attacks in the dataset, the experiment is run again after removing neptune from the set. Doing so results in a dramatic decrease in the percentage of the attacks detected.

5.3 Distance-based method

The former results give the impression that it will be very hard to detect anomalies in this dataset. A k-nearest neighbour-based algorithm is used to investigate this further. The results for running this algorithm on the whole test set computing the average distance to 10, 100 and 1000 closest neighbours and looking at the 2500 instances with the highest average are in table 5.1.

k	Attacks	Normal	% of attacks
10	1630	870	65.2%
100	1855	645	74.2%
1000	1912	588	76.5%

Table 5.1: Outliers in test set.

From this table it becomes clear that the number of neighbours k is very important. The percentage of normal values in the group of outliers is rather high. This might be because the number of attacks in the test set is extremely high. To determine if this is the case the experiment is done again with subsets of the test set. First subsets are created containing all normal instances and instances from one type of attack. When an attack type covers more than 1% of the subset, the number of instances of this type is brought back to 1%. The

results are in figure 5.6 to 5.11.

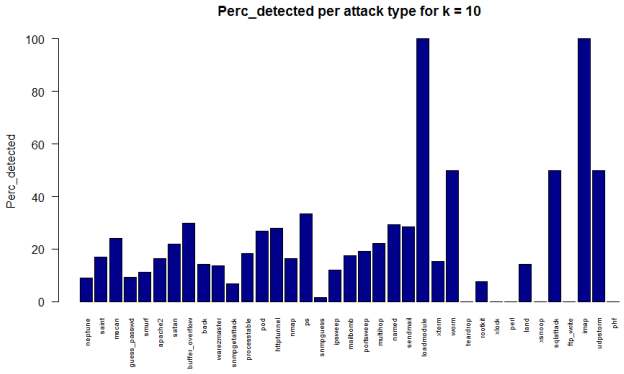

Figure 5.6: All attacks of one type, $k = 10$.

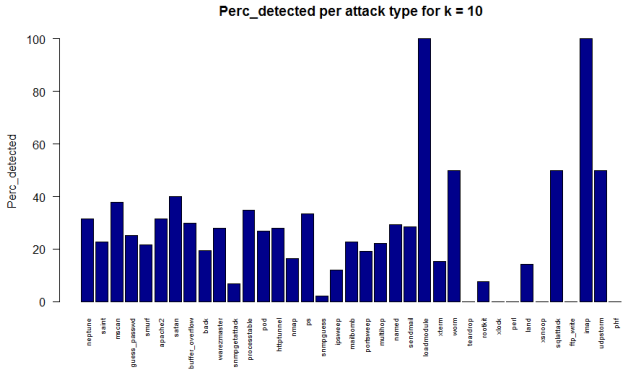

Figure 5.7: Maximum of 1% of attacks of one type, $k = 10$.

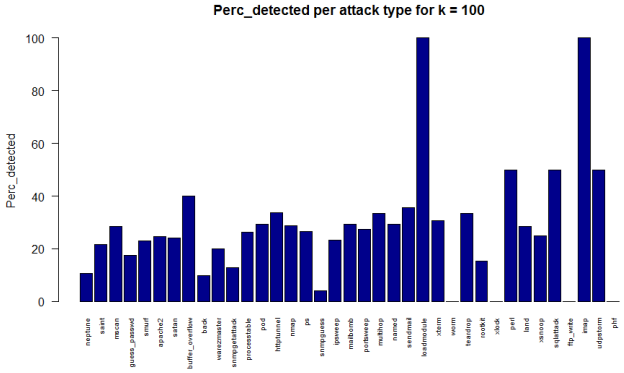

Figure 5.8: All attacks of one type, $k = 100$.

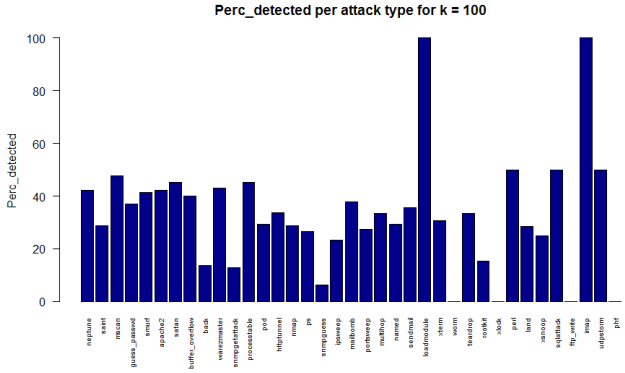

Figure 5.9: Maximum of 1% of attacks of one type, $k = 100$.

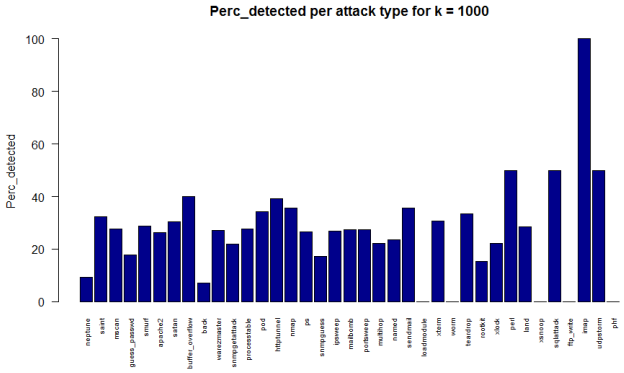

Figure 5.10: All attacks of one type, $k = 1000$.

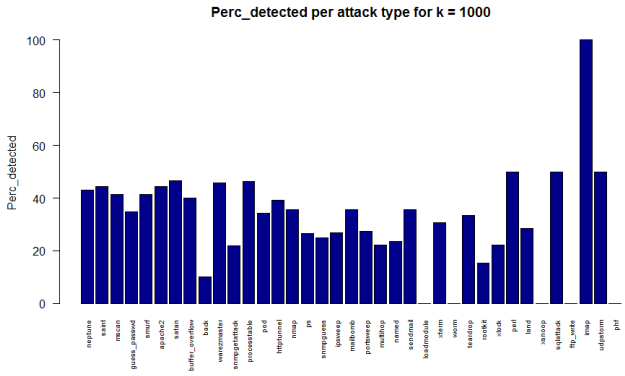

Figure 5.11: Maximum of 1% of attacks of one type, $k = 1000$.

Decreasing the number of attacks increases the detection rate. This is expected because when there is a large group of attacks of the same type these instances have more close neighbours which lowers the average distance. Since decreasing the number of attacks increases the detection rate the last type of subset created are subsets containing all normal values and a random sample of attacks that covers 1% of the subset. Figure 5.12 and figure 5.13 show some results.

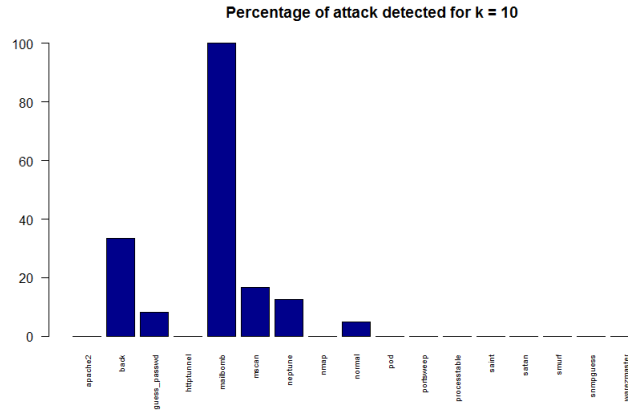

Figure 5.12: Sample 1.

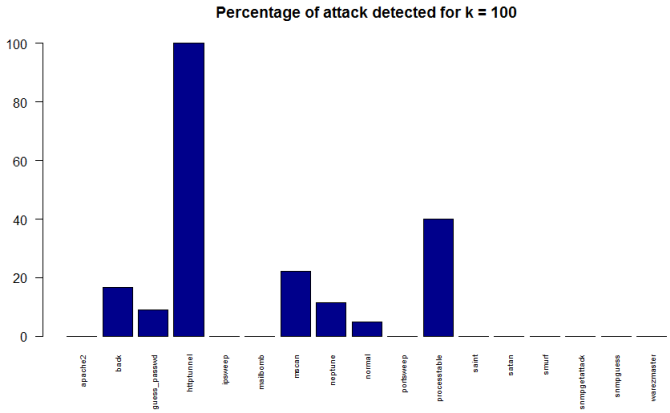

Figure 5.13: Sample 2.

The outliers cover 5% of the dataset whereas all attacks only cover 1%. Regardless of the number of neighbours k that is used, the percentage of attacks that are detected is very low for most attacks. It also highly depends on the sample which and how many attacks are detected.

Chapter 6

Conclusions

The aim of this thesis was to investigate the possibility of detecting cyber attacks using unsupervised anomaly detection. This is done by applying statistical methods and a distance based algorithm to the NSL-KDD dataset. The results show that this cannot be done reliably using the methods discussed in this paper. The true positive rate is far too low and the false positive rate way too high. There does not seem to be a pattern in the detected attacks. This is most likely caused by the fact that there is no clear distinction between the values of normal instances and attacks.

The difference between the accuracy of supervised methods and the unsupervised methods used in this thesis is dramatic. Even if it would be possible to reach a true positive rate around 90% there would be too many false alarms. In future research other methods can be tried for this and other datasets. It might however be more useful to look at contextual anomalies.

Appendix A

createstackedbar.R

```
#Create stacked bar chart for each attribute showing the
#distribution of attacks and normals
#Parameters have to be changed depending on the attribute
#If necessary uncomment the code for the right type of attribute
#Comment out the part of the code for the wrong type of attribute.

library('pracma')

train <- read.csv("KDDTrain+.txt")
train <- train[-43] #remove number attribute
attack <- train[42] != 'normal' #boolean vector with true for attack, false for normal

#For numeric attributes:
attribute <- train[1] #Select the attribute
#Change the range and number of bars if necessary:
s <- round(seq(0, max(attribute), length.out = 100), digits = 0)
n <- length(s)

normal <- mat.or.vec(n,1)
attacks <- mat.or.vec(n,1)

for(i in 1:n){ #Choose which bars should be plotted
  normal[i] <- sum(attribute >= s[i] & attribute < s[i+1] & attack == FALSE)
```

```
  attacks[i] <- sum(attribute >= s[i] & attribute < s[i+1] & attack == TRUE)
}

# For categorical and binary attributes:
# attribute <- train[2] #Select the attribute
# s <- as.matrix(unique(attribute)[1])
# n <- nrow(unique(attribute))
#
# normal <- mat.or.vec(n,1)
# attacks <- mat.or.vec(n,1)
#
# for(i in 1:n){ #Choose which bars should be plotted
#   normal[i] <- sum(attribute == s[i] & attack == FALSE)
#   attacks[i] <- sum(attribute == s[i] & attack == TRUE)
# }

#Create plot
counts <- rbind(attacks, normal)
barplot(counts, col=c('darkblue', 'red'), main = colnames(attribute),
        legend = rownames(counts), names.arg=s, las=2, cex.names=0.6)
```

Appendix B

countoutliers.R

```
#For all attributes calculate the percentage of attacks
#and normals in the top 5% and 10% of least frequent values
#For all pairs of attributes count the percentage of attacks
#and normals in the top 10% and 25% of least frequent values

#prepare data
train <- read.csv("KDDTrain+.txt")
train <- train[-43] #remove number attribute
attack <- train[42] != 'normal' #boolean vector: true for attack, false for normal
colnames(attack) <- 'attack'
train <- cbind(train, attack)
train <- train[-20] #attribute has only one value
forplot <- matrix(data = NA, nrow = 40, ncol = 2)

sink(file = 'outlier-count-10p.txt', append=TRUE)
percentage <- 90 #percentage covered by most frequent values
cat("\n \n Percentage of data with least frequent values is min ", percentage, sep="")
#number of outliers per attributes least frequent values
for(i in 1:40){
  #most frequent values:
  highestfreq <- as.data.frame(sort(table(train[i]), decreasing = TRUE))
  sum <- highestfreq$Freq[1] #number of observations with most frequent value
  k <- 1
```

```
  #if the most frequent value does not cover x% of observations
  while(sum < (length(attack)/100*percentage)){
    k <- k + 1
    sum <- sum + highestfreq$Freq[k]
  }
  val <- as.vector(highestfreq$Var1)[1:k]
  sub <- subset(train, !(as.matrix(train[i]) %in% val),
                select = c(colnames(train[i]), colnames(train[42])))
  numnor <- table(sub$attack)[1] #number of normals
  numatt <- table(sub$attack)[2] #number of attacks
  numtot <- dim(sub)[1] #total observations
  pertot <- numtot/length((attack))*100 #percentage of observations
  peratt <- numatt/numtot*100 #percentage of attacks
  pernor <- numnor/numtot*100 #percentage of normals

  cat("\n \n attribute is ", colnames(train[i]), sep="")
  cat("\n percentage is ", pertot, sep="")
  cat("\n number of total is ", numtot, sep="")
  cat("\n number of attacks is ", numatt, sep="")
  cat("\n number of normal is ", numnor, sep="")
  cat("\n percentage of attacks is ", peratt, sep="")
  cat("\n percentage of normal is ", pernor, sep="")

  forplot[i, 2] <- colnames(train[i])
  forplot[i, 1] <- peratt
  forplot[is.na(forplot)] <- 0
}
#make plot
par(mar = c(9, 3, 4, 2))
barplot(sapply(forplot[, 1], as.numeric), col = "darkblue", names.arg = forplot[, 2],
        main = "Percentage of attacks for 10% least frequent values",
        las = 2, cex.names = 0.7)
sink()

#Outliers per pair of attributes
```

```
percentage <- 75 #percentage covered by most frequent values
countMostlyAtt <- 0 #number of pairs with mostly attacks as outliers
countMostlyNor <- 0 #number of pairs with mostly normals as outliers

sink(file = '2-attr-outlier-count-25p.txt', append=TRUE)
cat("\n \n Percentage of data with most frequent values is min ", percentage, sep="")
for(i in 1:40){
  #outliers for attribute 1
  highestfreq1 <- as.data.frame(sort(table(train[i]), decreasing = TRUE))
  sum1 <- highestfreq1$Freq[1] #number of observations with the most frequent value
  k <- 1
  #if the most frequent value does not cover x% of observations:
  while(sum1 < (length(attack)/100*percentage)){
    k <- k + 1
    sum1 <- sum1 + highestfreq1$Freq[k]
  }

  #outliers for attribute 2
  for(j in 1:40){
    val1 <- as.vector(highestfreq1$Var1)[1:k]
    sub1 <- subset(train, !(as.matrix(train[i]) %in% val1),
                   select = c(colnames(train[i]), colnames(train[j]),
                              colnames(train[42]), colnames(train[41])))
    highestfreq2 <- as.data.frame(sort(table(train[j]), decreasing = TRUE))
    sum2 <- highestfreq2$Freq[1] #number of observations with the most frequent value
    k <- 1
    #if the most frequent value does not cover x% of observations
    while(sum2 < (length(attack)/100*percentage)){
      k <- k + 1
      sum2 <- sum2 + highestfreq2$Freq[k]
    }
    val2 <- as.vector(highestfreq2$Var1)[1:k]
    sub2 <- subset(sub1, !(as.matrix(sub1[2]) %in% val2),
                   select = c(colnames(train[i]), colnames(train[j]),
                              colnames(train[42]), colnames(train[41])))
```

```
    numnor <- table(sub2$attack)[1]
    numatt <- table(sub2$attack)[2]
    numtot <- dim(sub2)[1]
    pertot <- numtot/length((attack))*100
    peratt <- numatt/numtot*100
    pernor <- numnor/numtot*100
    attacktypes <- table(sub2[4])
    #if more attacks than normal outliers:
    if(numtot > 0 && !is.na(peratt) && (peratt > pernor || is.na(pernor))){
      countMostlyAtt <- countMostlyAtt + 1
    }
    #if more normal than attack outliers
    if (numtot > 0 && !is.na(pernor) && (pernor > peratt || is.na(peratt))){
      countMostlyNor <- countMostlyNor + 1
    }
    if(pertot > 1){ #found minimal of 1% outliers
      cat("\n \n attribute 1 is ", colnames(train[i]), sep="")
      cat("\n attribute 2 is ", colnames(train[j]), sep="")
      cat("\n percentage is ", pertot, sep="")
      cat("\n number of total is ", numtot, sep="")
      cat("\n number of attacks is ", numatt, sep="")
      cat("\n number of normal is ", numnor, sep="")
      cat("\n percentage of attacks is ", peratt, sep="")
      cat("\n percentage of normal is ", pernor, sep="")
      print(attacktypes)
    }
  }
}
cat("\n \n number of attributes with most attack outliers ", countMostlyAtt, sep="")
cat("\n number of attributes with most normal outliers ", countMostlyNor, sep="")
sink()
```

Appendix C

randomforest.R

```
#Make predictions for each attribute in the test set
#by building a random forest on other 40 attributes
#using the training set
#For each prediction calculate z-score and IQR
#for all instances and detect outliers

#The categorical attribute service had too many unique
#values for random forest. The least frequent values
#are grouped together to bring back the number of
#values to 53

library('randomForest')
library('EMCluster')

#Read data
train <- read.csv("Train-anomaly-ser.txt")
test <- read.csv("Test-anomaly-ser.txt")
testtype <- read.csv("Test-attype.txt")
testtype <- testtype[42] #attack types of test set
teAtt <- test[42] #anomaly or normal
train <- train[-42]
test <- test[-42]
```

```
pz <- c()

#calculate zscore
calc_zscore <- function(predictions){
  pmean <- mean(predictions)
  psd <- sd(predictions)*sqrt((length(predictions)-1)/(length(predictions)))
  scores <- (predictions-pmean)/psd
  pz <- cbind(pz, scores)
  outz <- (scores < -3) | (scores > 3)
  return(outz)
}

#calculate iqr
calc_iqr <- function(predictions){
  iqra <- IQR(predictions)
  quar <- quantile(predictions)
  lfence <- quar[2] - 1.5*iqra
  ufence <- quar[4] + 1.5*iqra
  outiqr <- (predictions < lfence) | (predictions > ufence)
  return(outiqr)
}

#calculate true and false positive rates for z-score and iqr and print to file
calc_rates <- function(outliers, filename, train, i){
  tpz <- sum(outliers[1] == TRUE & outliers[3] == 'anomaly')
  fpz <- sum(outliers[1] == TRUE & outliers[3] == 'normal')
  tnz <- sum(outliers[1] == FALSE & outliers[3] == 'normal')
  fnz <- sum(outliers[1] == FALSE & outliers[3] == 'anomaly')
  tprz <- tpz/(tpz + fnz)
  fprz <- fpz/(fpz + tnz)

  tpiqr <- sum(outliers[2] == TRUE & outliers[3] == 'anomaly')
  fpiqr <- sum(outliers[2] == TRUE & outliers[3] == 'normal')
  tniqr <- sum(outliers[2] == FALSE & outliers[3] == 'normal')
  fniqr <- sum(outliers[2] == FALSE & outliers[3] == 'anomaly')
```

```
  tpriqr <- tpiqr/(tpiqr + fniqr)
  fpriqr <- fpiqr/(fpiqr + tniqr)

  sink(filename, append=TRUE)
  cat("\n \n attribute is ", colnames(train[i]), sep="")
  cat("\n Z-score true positive rate: ", tprz, sep="")
  cat("\n Z-score false positive rate: ", fprz, sep="")
  cat("\n IQR true positive rate: ", tpriqr, sep="")
  cat("\n IQR false positive rate: ", fpriqr, sep="")
  sink()
}

#Random forest
for(i in 1:41){
  trPred <- train[-i]
  trResp <- as.matrix(train[i])
  tePred <- test[-i]
  teResp <- as.matrix(test[i])

  if(is.numeric(trResp[1])){
    set.seed(71)
    rf <- randomForest(x=trPred, y=trResp,
                       xtest=tePred, ytest=teResp, ntree=50, importance=TRUE)

    predictions <- rf$test$predicted - teResp

    if(is.numeric(predictions[1])){
      outz <- calc_zscore(predictions)
      outiqr <- calc_iqr(predictions)
      outliers <- cbind(outz, outiqr, teAtt, testtype)
      colnames(outliers) <- c('zscore', 'IQR', 'anomaly', 'attack type')

      if(is.numeric(trResp[1])){
        filename <- paste(colnames(train[i]), "-outliers.txt", sep="")
        write.csv(outliers, file=filename)
```

```
        calc_rates(outliers, "z-iqr-data.txt", train, i)
      }
    }
  }
}
```

Appendix D

nearestneighbour.R

```
# Find the top n data points whose average distance
# to the k nearest neighbours is greatest
# This is done for:
# The whole test set
# Subsets containing one type of attack and normal instances
# Subsets consisting of 1% of attacks and 99% of normals

library("kknn")
library("cluster")
library("class")

#Load data
test <- read.csv("KDDTest+.txt")

#Preprocess data
test <- test[-43] #remove number attribute
testAttack <- test[42] != 'normal' #boolean vector: attack is true
colnames(testAttack) <- 'attack'
test <- cbind(test, testAttack)

norTest <- test[1:41]

#normalize attribute x
```

```
normalize <- function(x){
  n <- x - min(x)
  d <- max(x)-min(x)
  return(n/d)
}

#normalize dataset
for(i in 1:41){
  if(is.numeric(as.matrix(test[i])) && (max(test[i]) > 1 || min(test[i]) < 0)){
    norTest[i] <- normalize(test[i])
  }
}

dissimilarity <- daisy(norTest, metric = 'gower',
                       type = list(symm = c(7, 12, 14, 21, 22)))
n <- attr(dissimilarity, 'Size') #number of observations
k <- 10 #k nearest neighbours
top <- 2500 #number of data points with greatest average distance
neighbours <- matrix(data = NA, nrow = n, ncol = 4)

#calculate the average distance between i and k nearest neighbours
for(i in 1:n){
   dist <- {}
   numbers <- {}
   for(j in 1:n){
     if(j != i){
      #distance between i and j:
      dist <- append(dist, dissimilarity[n*(i-1) - i*(i-1)/2 + j-i])
      numbers <- append(numbers, j) #row index
     }
   }
   distances <- as.data.frame(cbind(numbers, dist)) #distances between i and others
   ord <- as.matrix(distances[order(distances$dist), ]) #ordered list
   neighbours[i, 1] <- i
   #average distance between i and k nearest neighbours:
```

```
    neighbours[i, 2] <- mean(ord[1:k, 2])
    neighbours[i, 3] <- mean(ord[1:100, 2]) #k = 100
    neighbours[i, 4] <- mean(ord[1:1000, 2]) #k = 1000
}

neighbours <- as.data.frame(neighbours)
sortedk <- as.matrix(neighbours[order(neighbours$V2, decreasing = TRUE), ])
sorted100 <- as.matrix(neighbours[order(neighbours$V3, decreasing = TRUE), ])
sorted1000 <- as.matrix(neighbours[order(neighbours$V4, decreasing = TRUE), ])

sink("outliers-nn.txt", append = TRUE)
print(sortedk[1:top])
outliersk <- norTest[sortedk[1:top], 42:43]
print(summary(outliersk))

print(sorted100[1:top])
outliers100 <- norTest[sorted100[1:top], 42:43]
print(summary(outliers100))

print(sorted1000[1:top])
outliers1000 <- norTest[sorted1000[1:top], 42:43]
print(summary(outliers1000))
sink()

#Subsets containing one type of attack and normal instances
av <- cbind(norTest[1:22543, 42:43], neighbours[, 2:5])
attacktypes <- unique(norTest$attack_type) #attack types
attacktypes <- attacktypes[-2] #delete 'normal'
normal <- subset(av, attack_type == 'normal')
p <- 5 #percentage of data considered outliers
top <- floor(dim(av)[1]/100*p) #number of outliers
bar <- matrix(data = NA, nrow = 37, ncol = 4)
colnames(bar) <- c("attack_type", "perc_detected",
                   "perc_in_data", "perc_detected/perc_in_data")
```

```
sink("nnperattack1000k-1p.txt", append=TRUE)
cat("\n K = 1000")
for(i in 1:37){
  att <- sapply(attacktypes[i], as.character)
  natt <- sum(av$attack_type == att) #number of attacks of this type
  perc <- natt/dim(av)[1]*100

  if(perc > 1){ #attack covers more than 1% of data
   sub <- rbind(normal, subset(av, attack_type == att)[1:(floor(top/p)), ])
   natt <- floor(top/p)
  }
  else{
    sub <- rbind(normal, subset(av, attack_type == att))
  }
  sortsub <- as.matrix(sub[order(sub$V4, decreasing = TRUE), ])
  outliers <- sortsub[1:top, 1]

  bar[i, 1] <- att
  bar[i, 2] <- sum(outliers == att)/natt*100
  bar[i, 3] <- perc
  bar[i, 4] <- (sum(outliers == att)/natt*100)/(perc)

  cat("\n \n Attack type is ", att, sep="")
  cat("\n Number of attacks of this type in the dataset is ", natt, sep = "")
  cat("\n Percentage of this type in the dataset is ", natt/dim(sub)[1]*100, sep = "")
  cat("\n Number of attack detected ", sum(outliers == att), sep = "")
  cat("\n Percentage of attack detected ", sum(outliers == att)/natt*100, sep = "")
  cat("\n Summary of detected records: ")
  print(table(outliers))
}
sink()

barplot(sapply(bar[, 2], as.numeric), col = 'darkblue',  names.arg = bar[, 1],
        main = "Perc detected per attack type", ylab = "Perc detected", las = 2)
```

```
#Subsets consisting of 1% of attacks and 99% of normals
allatt <- subset(av, attack_type != 'normal')
pdf(file = "nnrsamp-outliers.pdf", onefile = TRUE)
for(i in 1:20){ #number of samples
  #create subset
  na <- floor(dim(normal)[1]/99)
  sam <- allatt[sample(nrow(allatt), size = na), ]
  sam <- rbind(normal, sam)
  na <- floor(dim(sam)[1]/100*p)

  #Find outliers
  sortsam <- as.matrix(sam[order(sam$V4, decreasing = TRUE), ]) #k = 1000
  outliers <- sortsam[1:na, 1]
  x <- as.data.frame(table(droplevels(sam$attack_type)))
  colnames(x) <- c('attack', 'in_sample')
  y <- as.data.frame(table(outliers))
  colnames(y) <- c('attack', 'detected')
  xy <- merge(x, y, by = intersect(names(x), names(y)), all.x = TRUE)
  xy[is.na(xy)] <- 0
  xy <- cbind(xy, xy$detected/na*100)
  xy <- cbind(xy, xy$detected/xy$in_sample*100)

  sink("nnrandomsamples-top5p.txt", append = TRUE)
  cat("\n \n Sample number: ", i, sep = "")
  cat("\n")
  print(xy)
  sink()

  barplot(xy[, 4], names.arg = xy[, 1], col = "darkblue",
          main = "Distribution of outliers", las = 2, cex.names = 0.6)
  barplot(xy[, 5], names.arg = xy[, 1], col = "darkblue",
          main = "Percentage of attack detected", las = 2, cex.names = 0.6)
}
dev.off()
graphics.off()
```

Bibliography

[Axe] Stefan Axelsson. The base-rate fallacy and its implications for the difficulty of intrusion detection. Department of Computer Engineering, Chalmers University of Technology.

[BBK15] Monowar H. Bhuyan, Dhruba K. Bhattacharyya, and Jugal K. Kalita. Towards generating real-life datasets for network intrusion detection. *International Journal of Network Security*, 17(6):675–693, 2015.

[CBK07] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. Technical Report TR 07-017, Department of Computer Science and Engineering, University of Minnesota, August 2007.

[DS15] L. Dhanabal and S.P. Shantharajah. A study on nsl-kdd dataset for intrusion detection system based on classification algorithms. *International Journal of Advanced Research in Computer and Communication Engineering*, 4(6):446–452, June 2015.

[EAP+] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Sal Stolfo. A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. Department of Computer Science, Columbia University.

[IoC99] Information and Computer Science University of California. Kdd cup 1999 data, 1999. Available on: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[Lin13] Dong Lin. Network intrusion detection and mitigation against denial of service attack. Technical Report MS-CIS-13-04, Department of Computer and Information Science, University of Pennsylvania, January 2013.

[MRS+16] Martin Maechler, Peter Rousseeuw, Anja Struyf, Mia Hubert, and Kurt Hornik. *cluster: Cluster Analysis Basics and Extensions*, 2016. R package version 2.0.4.

[oe15] Information Security Center of eXcellence. Unb iscx nsl-kdd dataset, 2015. Available on: http://www.unb.ca/research/iscx/dataset/iscx-NSL-KDD-dataset.html.

[Ope] OpenStax. The standard normal distribution. Available on: http://cnx.org/contents/KgL8DwG_@4/The-Standard-Normal-Distributi.

[PES] Leonid Portnoy, Eleazar Eskin, and Sal Stolfo. Intrusion detection with unlabeled data using clustering. Department of Computer Science, Columbia University.

[RM13] S. Revathi and A. Malathi. A detailed analysis on nsl-kdd dataset using various machine learning techniques for intrusion detection. *International Journal of Engineering Research and Technology*, 2(12):1848–1853, December 2013.

[SHR] Anja Struyf, Mia Hubert, and Peter Rousseeuw. Dissimilarity matrix calculation. Available on: http://stat.ethz.ch/R-manual/R-devel/library/cluster/html/daisy.html.

[SoMLL] Cyber Systems and Technology Group of MIT Lincoln Laboratory. Darpa intrusion detection data sets. Available on: http://www.ll.mit.edu/ideval/data/.