# Machine Learning Techniques for Intrusion Detection Systems

Bachelor Thesis

**Authors**: Andreas Heidelbach Engly [s170303] and Anton Ruby Larsen [s174356]
**Supervisor**: Assistant Professor Dr. Weizhi Meng

Department of Applied Mathematics and Computer Science
Technical University of Denmark

May 14, 2023

# Abstract

In this thesis, we propose an ensemble of two neural networks and a gradient-boosting machine trained on a feature-selected and imbalance-corrected version of NSL-KDD. It is evaluated on the corresponding test data that comes separately with the NSL-KDD. We conducted our experiments on Random Forests, Neural Networks, and Gradient-Boosting Machines on the raw data set, and then it was repeated on the preprocessed versions. By observing the individual advantages of the classifiers, we combined them into an ensemble with the goal of benefiting from each of these. The proposed model improved significantly from the preprocessed data compared to a similar model trained on raw data: 10% points in overall accuracy and 24% points in overall F1-score. Despite the improvements, we realise it cannot be used as a stand-alone system. We therefore suggest an intrusion detection system consisting of a misuse-based intrusion detection filter combined with our proposed model. The misuse filter will be able to detect all known attacks with a very little error. Our proposed model would then be able to seamlessly work in the background and detect unknown attack types with a false alarm rate under 1%.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

While companies, organizations and individuals benefit greatly from increased digitisation and interconnection, so do the criminals. Systems have grown in size and complexity, and it has made it hard to maintain high security standards. Cybersecurity Ventures, a world leading researcher in global cyber economy, predicts the annual loss from cybercrime to exceed 6 trillion dollars by 2021 [4].

In the early days hacking was reserved for the few. Nowadays with Hacking-as-a-Service [HaaS], the number of entities that pose a threat has increased significantly. It is no longer required to be an expert when one can acquire sophisticated tools for doing the tasks. The hackers have all sorts of motivations. It could be political, financial, or even intellectual. It is unthinkable that an organization or a company will not at some point fall victim to a hacking attempt.

The difficulties have proven themselves during the COVID-19 crisis. The crisis made governments take swift actions, and companies and organizations have to adapt to operating remotely within days. The unusual network traffic has made it hard to distinguish the good from the bad. Though COVID-19 is no ordinary situation, it certainly cements the need for scalable, versatile, and intelligent solutions. Anomaly-based Intrusion Detection Systems cope especially well with these issues. In this thesis, we seek to explore how well conventional machine learning methods can be used in Anomaly-based Intrusion Detection, and how these can be improved by different preprocessing methods.

Our main contributions are:

1. An overview of imbalance-correction with SMOTE.
2. An overview of correlation-based, fast-correlation-based, and consistency-based feature selection.
3. A detailed explanation of Neural Networks, Random Forests, and Gradient-Boosting Machines. In addition, how combining these can improve the ability to generalize.
4. An analysis of how the models perform when trained on the preprocessed data sets obtained from imbalance-correction and feature selection on NSL-KDD.

We start by creating an imbalance-corrected data set with an algorithm named SMOTE. Then we perform feature selection on the original data with three different methods: correlation-based, fast-correlation-based, and consistency-based feature selection. The data sets obtained from the previous steps are then used to train an ensemble of neural networks, a random forest, a gradient-boosting machine, and an ensemble of neural networks and a gradient-boosting machine. These results will be discussed in the context of a real framework, where we balance the cost and benefits of deploying a system using these models.

# 2 Intrusion Detection Systems

The field of network security involves various measures and serves to counteract undesirable activity. While measures such as firewalls wants to assure that external and internal communication happens in accordance with certain sets of predefined security policies, they have a significant drawback: protection against the unknown. It is unthinkable that even the brightest minds will foresee all sorts of malignant creativity, and therefore we need a system that can effectively detect a wide range of intrusion types. Such a system is known as Intrusion Detection System and is commonly abbreviated IDS.

IDSs are grouped into network-based IDS [NIDS] and host-based [HIDS]. Usually NIDSs analyse network-based traffic by examining headers and possibly payload, while the HIDSs monitor log-files, system calls and more on a protected host [5].

An IDS can be extended to take action on raised alarms. This is commonly referred to as an Intrusion Prevention Systems [IPS]. The actions must be carefully chosen, since an IDS system will raise false alarms. This will impact the availability of certain services, even when there is no need to. Choosing the actions in order to extend the IDS is beyond the scope of this thesis, and we will confine ourselves to detection only.

## 2.1 Detection Approaches

An IDS can be divided into three subcategories based on the method of detection [6].

The first is misuse-based detection, where the system tries to match monitored traffic against a signature-database. This is highly effective for known attacks, but is incapable of detecting attacks with a new structure.

The second is a specification-based detection, where the detection model monitors if system activity deviate from the way normal activity is defined. Since it is not possible to comprehensively define normal activity when taking unknown attacks into consideration, it does not perform well on unknown attack patterns.

The third is anomaly-based detection, which tries to solve the shortcomings for unknown attacks of the preceding detection methods. The idea is to obtain some degree of independence from expert-knowledge such that underlying structures can be captured.

The usage of machine-learning-techniques belongs to the third category, anomaly-based detection. The methods explored in the rest of this thesis seeks to evaluate such methods by classifying existing attacks into 4 attack-types. The goal is to be able to capture existing and new attacks in each of these categories [6]. We refer to Appendix A for further information.

## 2.2 Data Collection

A way to gather information about network traffic is to capture data being transferred to, from, and across networks. This is done by deploying sensors at points in the network infrastructure where the data passes by. Since it is hard to analyse single packages, they are usually combined in order to infer anything useful. This provides a context to the analysis, and it can improve the ability to detect anomalies [7].

It can place a heavy load on the system to send entire packages to the processing engines, where the classification takes place. For this reason, sensors usually forward a selected set of features. This has the advantage that the load is distributed to several units across the network [7]. Methods for selecting a subset of features that maintain a high level of information is explored in a later section.

## 2.3 System Architecture

A common way of organizing IDSs is in a centralized way. This means that a single detection engine receives all the captured network traffic from the sensors [1]. In the context of anomaly-based detection it means the engine will receive captured traffic from sensors across the network and then try to classify it.

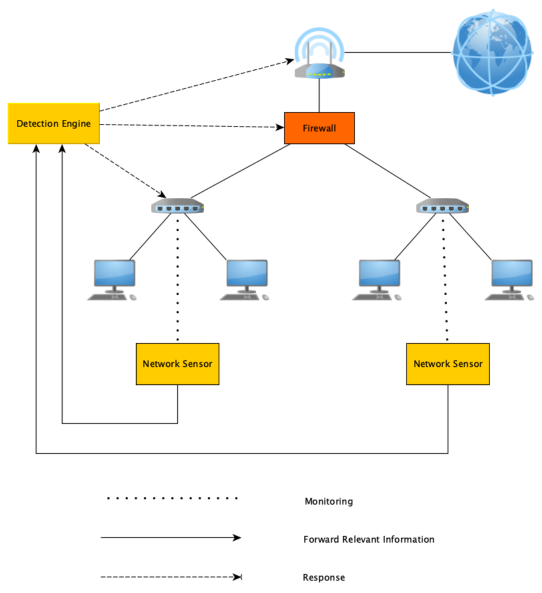

Figure 1: Overview of a generic centralized architecture drawn in yEd [1]

A problem with this architecture is the high load on the processing engine, which can lead to dropped packages. Furthermore, it lacks fault tolerance due to no redundancy. A better solution, though more expensive and complex, is a distributed architecture in which processing engines are placed locally.

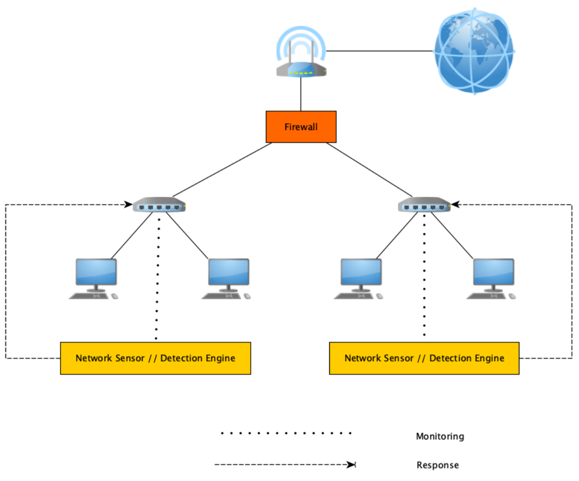
Firewall
Network Sensor // Detection Engine
Network Sensor // Detection Engine
Monitoring
Response

Figure 2: Overview of a generic distributed architecture drawn in yEd [1]

The above mentioned are generic structures, and they come with many modifications. However, for the purpose of our thesis this provides a sufficient overview.

# 3 Data Sets for Benchmarking

In our thesis we train all the later introduced machine learning models with the network intrusion data set NSL-KDD99. This data set is an improved version of the KDD99 data set that we will start by describing.

## 3.1 KDD99

The KDD99 is a network intrusion data set developed by MIT for the international data mining competition in 1999. As it is described in [8], KDD99 is based on the DARPA98 evaluation program which was a 7 week long simulation. The simulation was based on several months of observed data traffic obtained from a number of American air force bases. The simulation, as described in [2], therefore included several different devices running different operating systems and services. The specific network topology can be seen in Figure 3.

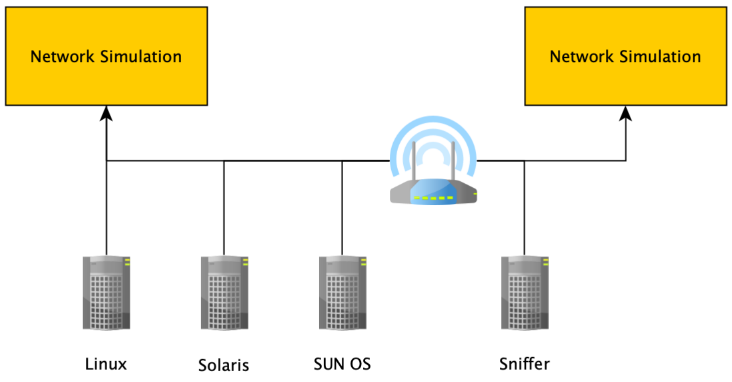

Figure 3: Illustration of the network topology drawn in yEd [2]

The 7 weeks of simulation produced approximately 4 GB of compressed tcpdumps. They were converted into 6.9 million connection consisting of 41 features, a specific attack type, and a more general attack group.

4.9 million of the connections constituted the training data of the KDD99 data set, and the rest of the connections was the test set.

The general attack groups are DoS, U2R, R2L, probing, and normal[1]. We will not describe all the specific attack types because we will only predict the more general groups in our thesis[2]. For the particular mapping we refer to Appendix C.

In [8] they describe the 41 features of each connection and states that the features are composed of 3 main groups, namely basic features, traffic features and content features.

[1]Where normal of course is not an attack type.
[2]For examples of each major attack group see Appendix D

**Basic features** contain information that can be retrieved directly from the IP/TCP-connection without any further processing.

**Traffic features** can be thought of as time related features. There are two subgroups within this feature: one describing a time interval of 2 second, and one describing a time interval of 100 connections. Each of these subgroups contains again two categories, "same host" and "same service" features. The "same host" features contain statistical operations on protocol behaviour, services, and etc. for connections in the given time interval and with the same destination host. The "same service" does the same but only for connections with the same service. The reason for the two time intervals is to catch both attacks that has a lot of connections in a short amount of time, and probing attacks that span over longer time intervals.

**Content features** examine data portions of each connection. This group of features is designed to be able to detect U2R and R2L attacks. These do not have a sequential pattern between the connections, but can be disclosed by e.g. failed login attempts.

### 3.1.1 Problems related to the KDD99 data set

The KDD99 data set has been severely criticised for a number of reasons. Some are inherited from the DARPA98 program and some are specific for the KDD99 data set. We will start taking a look on the problems related to the DARPA98 simulation.

Problems related to the DARPA98 program are described in [8] and are the following:

- The use of tcpdumps as traffic collector is criticised. Tcpdumps has a tendency to drop packages or overload in heavy traffic, and the authors behind DARPA98 do not provide any examination to check the possibility of dropped packages.
- The simulation of military networks required heavy privacy which resulted in no information about the background of the simulation.
- In [8] they point out a lack of precise descriptions of each attack type. For example, probing is not necessarily an attack, but first becomes an attack when the number of iteration exceeds some threshold.

All these problems related to the DARPA98 simulation are hard if not impossible to remediate. When that is said, KDD99 also has some problems on its own which are fixable. How these can be corrected will be discussed in section 3.2.

One of the most severe problems is the huge redundancy that is found both in the test and training set of KDD99. 78.05% of the training set and 75.15% of the test set is redundant leading to a bias towards frequent samples when used for training learning algorithms.

Another problem is the sheer size of KDD99. Due to the size of the data set researchers tend to pick random subsets and train learning algorithms on these. This makes the different results hard to compare due to different training and test data.

A last problem is the disproportionate amount of DoS attacks. 71% of all the data is two specific types of DoS attacks making cross validation impossible, because several folds will contain nothing but these two specific DoS attack types.

## 3.2 NSL-KDD99

The researchers behind NSL-KDD99 did several things in [8] to improve the KDD99 data set. Firstly they removed all redundant samples. Next they chose 7 of the most popular machine learning algorithms at that time and trained them on 3 randomly selected data sets from the KDD99 training data each containing 50,000 samples.

This resulted in 21 models they used to predict all data in the whole KDD99 data set. All samples where then polled into groups describing how many of the models had gotten that sample right. The succeeding five groups were made, 0-5, 6-10, 11-15, 16-20 and 21 learners predicted the sample correct. The samples were then split into test, and training data and the percentage each group composed was calculated.

Each group constituted a percentage of the whole KDD99 data set. To put emphasis on harder instances while also reducing the size of the space the authors of NSL-KDD99 took the inverse of each group and sampled at random this amount from the given group. This resulted in a more even ratio between attack types, and hence also an increased difficulty. Due to the increase in difficulty the performance of the different machine learning algorithms was also more spread out on the new data set.

In consequence of the above actions the data set decreased to a manageable size. This makes it possible to train learning algorithms on the whole data set, and hence possible to compare different projects.

The NSL-KDD99 is far from perfect. Simulation of a military network can hardly represent an every day public network. This is also mentioned by the researchers behind the NSL-KDD99 data set but they argue that as a benchmarking data set for machine learning algorithms in network detection it will work fine. Furthermore a lot of articles have been published using this data set, which provides a solid foundation for comparison. For these reasons, we chose the NSL-KDD99 data set.

# 4 Handling Imbalanced Data

When using imbalanced data for classification, it can lead to a large bias towards the majority class. This can be counteracted by applying what is known as sampling techniques.

Sampling techniques can be divided into two groups: under-sampling and over-sampling. When doing under-sampling we remove samples of the majority class to balance out the ratio between the classes. When having very imbalanced classes as in our data set[3], this is in general not a beneficial technique. To get a reasonable ratio between the minority and majority class we would need to remove a lot of information from the majority class we otherwise could have used for better classification.

Therefore we normally apply over-sampling where we upscale the minority class instead. The simplest way to do this is by sampling with replacement from the minority class until balance has been established between the classes. One problem with this method is when facing severe imbalances the up-scaled minority class would consist of several replications of the same data leading to overfitting of the minority class.

We will therefore look at the algorithm Synthetic Minority Over-sampling Technique [SMOTE].

## 4.1 Synthetic Minority Over-sampling Technique

Instead of sampling with replacement, SMOTE creates new synthetic samples of the minority class by use of the K-nearest neighbours [KNN] algorithm. For every sample in the minority class, SMOTE finds the KNN of minority samples and chooses $n$ of the neighbours at random, where $n$ depends on how much the minority class needs to be up-scaled. Then a random point on the straight line segment between the sample in focus and the chosen neighbour is generated as a new synthetic sample.

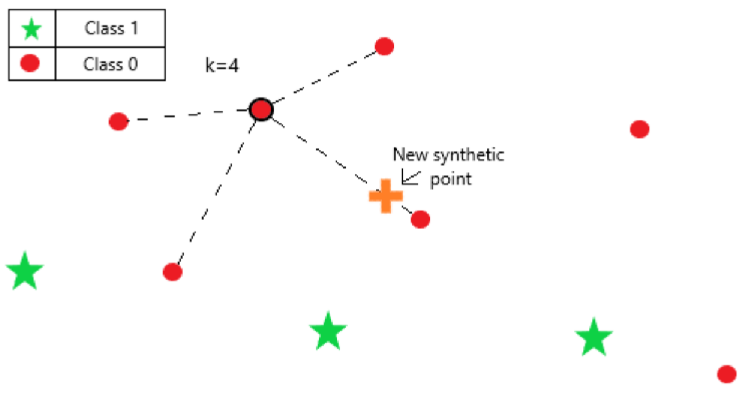

Figure 4: One synthetic sample is generated by use of SMOTE

Instead of just repeating the same samples we generate new nearby points assuming they are good candidates for potential new instances of the minority class. One problem we have to solve

[3]See Appendix B

to be able to use SMOTE for our data is the measure of distance between mixed data. To handle this we will introduce the metric Heterogeneous Value Difference Metric(HVDM).

## 4.2 Heterogeneous Value Difference Metric

This metric was introduced by [9] and handles both missing values, numeric and nominal data and is defined as

$$\text{HVDM}(x,y) = \sqrt{\sum_{i=1}^{m} d_{f_i}(x_{f_i}, y_{f_i})^2}$$

Where

$$d_{f_i}(x_{fi}, y_{fi}) = \begin{cases} 1 & \text{if } x_{fi} \text{ or } y_{fi} \text{ is missing} \\ \text{normalized vdm}_{fi}(x_{fi}, y_{fi}) & \text{if } f_i \text{ is nominal} \\ \text{normalized diff}_{fi}(x_{fi}, y_{fi}) & \text{if } f_i \text{ is numeric} \end{cases}$$

We see that nominal specific and numeric specific metric are used within the HVDM. The numeric distance, diff, is the euclidean metric defined as

$$\begin{aligned} \text{diff}_{fi}(x_{fi}, y_{fi}) =& \sqrt{(x_{fi} - y_{fi})^2} \\ =& |x_{fi} - y_{fi}| \end{aligned}$$

and we normalize it by dividing with 4 times the standard deviation of the feature resulting in

$$\text{normalized diff}_{fi}(x_{fi}, y_{fi}) = \frac{|x_{fi} - y_{fi}|}{4\sigma_{fi}}$$

The nominal distance is calculated using the metric Value Difference Metric [VDM], which is based on the conditional probability of seeing class c given sample x. The metric is defined as

$$\begin{aligned} \text{vdm}_{fi}(x_{fi}, y_{fi}) =& \sum_{c=1}^{C} \left| \frac{N_{f_i,x,c}}{N_{f_i,x}} - \frac{N_{f_i,y,c}}{N_{f_i,y}} \right| \\ =& \sum_{c=1}^{C} \left| P(c|x_{f_i}) - P(c|y_{f_i}) \right| \end{aligned}$$

The normalization of the VDM is given as

$$\text{normalized vdm}_{fi}(x_{fi}, y_{fi}) = \sqrt{\sum_{c=1}^{C} \left| P(c|x_{f_i}) - P(c|y_{f_i}) \right|^2}$$

## 4.3 Problems with SMOTE

While SMOTE is a very wide spread method to deal with imbalances in the data it also has its limitations. It assumes that the convex hull for the $k$ minority samples in a $k$ large neighbourhood is a realistic representation for the underlying distribution of the $k$ minority samples. This is not always given and especially when the dimensions grow large, SMOTE will suffer under the curse of dimensionality, which is explained in the next section. The reason for this is that sparsity will grow exponentially with the dimensions and therefore points of the minority class will have very large distances between them. With that being said, by use of the HVDM metric we are able to apply the SMOTE algorithm to our data without one hot encoding, and hence keeping the dimension fairly low. Thereby we reduce the risk of suffering under the curse of dimensionality.

Another point is that SMOTE does not discern hard instances of the minority class over easier. This problem has actually been solved by the AdaSyn algorithm introduced by [10]. The amount of extra minority points AdaSyn generates for a given neighbourhood depends on the amount of majority samples in the neighbourhood. The more majority samples in the neighbourhood, the harder it will be to differentiate the minority samples from the majority, and hence more must be synthesized.

While SMOTE is very computationally intensive, AdaSyn is even heavier. We tried to apply the algorithm to our data, but the algorithm required 117 GB of RAM to be used on NSL-KDD99, and hence we decided to use SMOTE instead.

# 5 Feature Selection

When we want to predict something by use of machine learning, we use data to estimate a model. This is true for all the different techniques in this domain, and therefore we want the data to represent what we want to predict as good as possible.

The data sets we are dealing with in the real world are always finite, and therefore we need to face what is known as the curse of dimensionality.

The curse of dimensionality shows that the number of regions in a space grows exponentially with the number of dimensions. This can be illustrated with Figure 5, which shows a 10 by 10 grid containing a marked 8 by 8 grid.

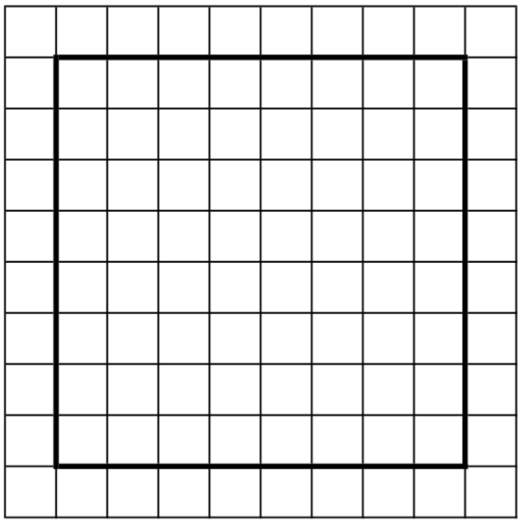

Figure 5: $10 \times 10$ grid

If we assume we have 100 uniformly distributed samples in the 2 dimensional space, 64% of the samples will lie within the 8 by 8 box. We then expand this space to 41 dimensions as we have in the NSL-KDD data set and keep the properties that all side lengths are 10 and the space consists of 1 by 1 discrete squares that contains one data point. We can calculate how much data we would need to conceptually describe 64% of the space.

$$\frac{64}{100} \cdot 10^{41} = 6.4 \cdot 10^{40}$$

This is way larger than the 125973 samples the NSL-KDD99 training set consists of, which means we will be working in a very sparse space. This is a problem, because the models we wish to train can get stuck in small clusters in the space that has nothing to do with our classes. We therefore want to reduce the number of dimensions and reduce this risk.

## 5.1 Information Theoretic Methods

The NSL-KDD99 data set consists of both numeric and nominal data, and we therefore need a framework that can handle this. For this reason, we introduce methods based on information theory because it handles nominal and numeric data quite well.

### 5.1.1 Entropy

In section 2.1 in [11] entropy is given as.

$$H(X) = -\sum_{x \in X} P(x) \cdot \log_2(P(x))$$

It describes how much "information" a random variable contains and to explain how it works we will consider a coin denoted C.

$$C = \begin{cases} P(heads) = \theta \\ P(tails) = 1 - \theta \end{cases}$$

We have the entropy of this coin given by

$$H(C) = -((1-\theta) \cdot \log_2(1-\theta) + \theta \cdot \log_2(\theta))$$

We plot the entropy of the coin as a function of $\theta$ in Figure 6 to get a understanding of how the entropy changes as the coin gets less and less fair.

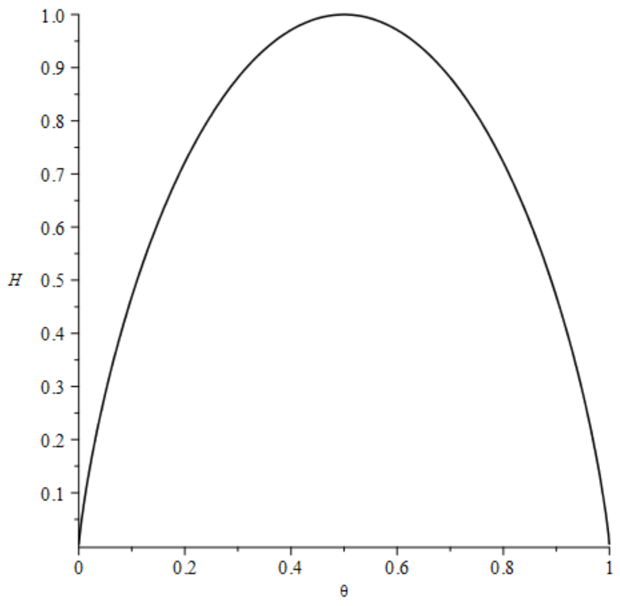

Figure 6: Entropy for the coin C as $\theta$ changes

We see that when $\theta = \frac{1}{2}$, $H(C) = 1$ meaning we need 1 "bit" to represent all information in the random variable C. This makes sense because if the coin comes out tails we can show it by 0 and heads by 1. If $\theta = 0$, the entropy is also 0 meaning we need no bits to represent the coin.

The reason for this is that all information is known before hand and no new information needs to be represented, i.e. no uncertainty.

One can think of this as an information theoretic counterpart to classic variance.

### 5.1.2 Information Gain

Because we want to predict Y from X we want some counterpart to the concept of covariance. This counterpart in information theory is called information gain, described in section 2.3 and 2.4 in [11], and is defined by.

$$\begin{aligned} I(X,Y) &= \sum_{x\in X}\sum_{y\in Y} P(x,y)\cdot \log_2\left(\frac{P(x,y)}{P(x)P(y)}\right) \\ &= H(X)+H(Y)-H(X,Y) \\ &= H(Y)-H(Y|X) \end{aligned}$$

So what we quantify is how much information knowing X tells us about Y.

To understand this better we will take an example of two coins. In the first case we will consider our coins to be fair and therefore exhibiting the distribution pictured in Figure 7.

| | | $C_2$ | |
|---|---|---|---|
| | | 0 | 1 |
| $C_1$ | 0 | 0.25 | 0.25 |
| | 1 | 0.25 | 0.25 |

Figure 7: Probability distribution for two fair coins

The information gain for the coins is

$$I(C_1,C_2) = H(C_1)+H(C_2)-H(C_1,C_2) = 1+1-2 = 0$$

We see that the information gain equals 0 because the coins are completely independent and therefore fair. We now take an example from the other extreme.

| | | $C_2$ | |
|---|---|---|---|
| | | 0 | 1 |
| $C_1$ | 0 | 0.5 | 0 |
| | 1 | 0 | 0.5 |

Figure 8: Probability distribution for two unfair coins

The information gain is now

$$I(C_1, C_2) = 0.5 \cdot \log_2 \frac{0.5}{0.5^2} + 0.5 \cdot \log_2 \frac{0.5}{0.5^2} = \log_2 2 = 1$$

It is seen that knowing one coin gives you the other one and hence the information gain is 1.

### 5.1.3 Information Gain Ratio

Information gain will be biased towards features with a lot of the same value because this will lead to a higher entropy as variance will be higher for higher spread in values. Therefore we will introduce the concept information gain ratio described in [12]. It corresponds to a information theoretical counterpart to correlation.

$$\begin{aligned} GR(X,Y) &= \frac{H(Y)}{H(X)} - \frac{\sum_{x \in X} \sum_{y \in Y} P(x,y) \cdot \log_2(\frac{P(x)}{P(x,y)})}{H(X)} \\ &= \frac{H(Y)}{H(X)} - \frac{H(Y|X)}{H(X)} \end{aligned}$$

We normalise the conditional entropy by the entropy of X which eliminates the bias information gain suffered from. One problem with this measure of correlation is that it is asymmetrical. This is problematic when we want to minimise redundancy internally in the feature space.

### 5.1.4 Symmetric Uncertainty

To solve the problem of asymmetricality we will introduce a symmetrical measure called symmetric uncertainty, described by [13].

$$SU(X,Y) = 2\frac{I(X,Y)}{H(X) + H(Y)}$$

This measure inherits both the property of measuring a counterpart to correlation and being symmetric.

### 5.1.5 Fast Correlation-Based Filter

Now we have established counterparts to the statistical second moments and will look at an algorithm for selecting a subset of features. The first algorithm we will look at is called Fast Correlation-Based Filter [FCBF] which was introduced by [13]. Our goal is to maximise the relevance of the input features with respect to the classes and minimise the redundancy between the input features.To meet these two objectives we will introduce a couple of concepts described in [13]. First we will define some subsets of the feature space.

We define $F$ as a set of features, $F_i$ as a specific feature and $S_i = F - F_i$ as the subset of the features $F$ without $F_i$

**C and F correlation:** We will establish a correlation between features called F-correlation and a correlation between features and classes called C-correlation.

$$C - correlation: \; SU_{i,Y} = SU(F_i, Y)$$
$$F - correlation: \; SU_{i,j} = SU(F_i, F_j)$$

**Approximate Markov blanket:** $F_j$ forms an approximate Markov blanket for $F_i$ if and only if

$$SU_{j,Y} \geq SU_{i,Y} \wedge SU_{i,j} \geq SU_{i,Y}$$

**Relevance:** A feature $F_i$ is relevant if

$$SU_{i,Y} \geq \gamma \;\wedge\; P(Y|F_i, S_i) \neq P(Y|S_i)$$

**Predominant features:** A relevant feature is predominant if and only if it does not have any approximate Markov blanket in the current set of features.

We now have a concept of relevance, redundancy by the approximate Markov blanket and a joined measure of both redundancy and relevance by the concept of a predominant feature. We will now write up the FCBF-algorithm utilising these concepts.

1. Select a predominant feature.

2. Remove all features for which it form an approximate Markov blanket.

3. Repeat 1. and 2. until only predominant features are left in the input space.

### 5.1.6 Correlation-Based Filter

A second method to maximise the correlation between the class set and the feature set meanwhile minimising the correlation within the features is introduced by [14]. Here the following equation is used to evaluate a subset of features.

$$M_S = \frac{k\bar{r}_{Yf}}{\sqrt{k + k(k-1)\bar{r}_{ff}}}$$

Where $M_s$ is the score for the subset of features $S$, and $k$ is the number of features in $S$, i.e. $|S|$. $\bar{r}_{Yf}$ is the average correlation between the class set $Y$ and the features in $S$, and $\bar{r}_{ff}$ is the average correlation within $S$.

We observe that the evaluation function is in-fact a modified version of Pearson's correlation coefficient where the predictiveness of S with respect to Y and normalised by the redundancy in S.

The correlation measure $r$ is the symmetrical uncertainty and to discretise the feature space the minimum description length is used. It is described in further detail in [15].

To find the optimal subset of all the features we use the search technique best-first search. For the best-first search we use the stopping criteria that when five consecutive fully expanded subsets show no improvement over the best found, we stop, as stated on page 70 in [14].

## 5.2 Consistency Methods

Another way to determine a subset of the feature space is consistency.

Consistency in statistics is when a population-sample grows, the estimator for the population distribution converge towards the true mean of the population. To translate this concept into an applicable method we will use methods introduced by [16] and [17].

Inconsistency will be measured by Liu's measure of inconsistency.

$$Inconsistency = \frac{number\ of\ inconsistent\ examples}{number\ of\ examples}$$

Where inconsistent and consistent examples are illustrated in Figure 9.

| | $x_1$ | $x_2$ | y |
|---|---|---|---|
| a | 1 | 1 | 1 |
| b | 1 | 1 | 0 |
| c | 1 | 0 | 0 |

Figure 9: Table of consistent and inconsistent samples

In Figure 9 we see that sample a and b are inconsistent because all their features are identical, but the class is different for the two. The samples a and c do not have identical features and can therefore have different values for the class without getting inconsistent. Because inconsistency always lies between 0 and 1, consistency is defined by:

$$Consistency = 1 - Inconsistency$$

One advantageous property of our consistency measure is that it is monotonic. This means that if we have some subsets $\{S_0, S_1, ..., S_n\}$ of the feature space, and we denote the measure of consistency by U it holds that:

$$if\ S_0 \subset S_1 \subset ... \subset S_n \implies U(S_0) \leq U(S_1) \leq ... \leq U(S_n)$$

This gives us the ability to apply the search technique best-first search for the optimal subset among the $2^m$ subsets of features, where $m$ is the cardinality of our feature set. If we did not apply such a technique, it would become an infeasible task to find the optimal subset.

## 5.3 Summary

The above described methods based on information theory where chosen due to information theoretic methods ability to both handle nominal and numeric data. The FCBF method and the CFS method have different stopping criteria but they both seek the same goal. They both seek to maximise correlation between the input features and the classes as well as minimise the correlation between the input features themselves.

We also wanted to have a method that was not based on information theory. We therefore introduced a method based on consistency. This concept is different from methods based on classic correlation or information theoretic correlation but the goal is the same. The consistency method also seek to maximise relevance of the input features by reducing redundancy between them and maximise descriptive power regarding the output classes.

These three methods will be evaluated on the NSL-KDD99 data set in section 9.

# 6 Model Theory

## 6.1 Tree-based models

Models solving the problem of classification are one of two major groups in supervised machine learning, where the other is regression. People have come up with a wide variety of methods to solve the problem and one major group of techniques for this problem are tree-based models. For many of these the base learner is the decision tree.

### 6.1.1 Decision Trees

Decision trees are build of nodes and leaves as illustrated in Figure 10

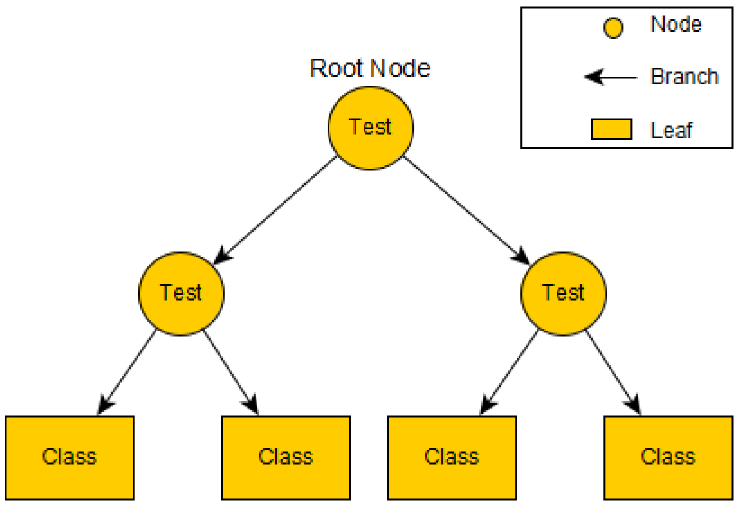

Figure 10: Example of a decision tree

We input a sample at the top of the tree and in each node one feature of the sample will be tested and decide where to go next in the tree. Eventually the sample will end up in a leaf-node determining the class of the sample.

The most complicated part of building a decision tree is determining the best feature to test at each split in the tree. To tackle this problem many algorithms have been developed. We will consider the algorithm CART for splitting described in section 1.2 in [18].

**Splitting with CART** : When we want to decide which feature makes the best split of the data at a specific location $m_p$, we consider an impurity function $i(m)$. Our goal is to minimise the impurity when splitting and we must therefore select the feature that minimises the following expression.

$$\Delta i(m) = i(m_p) - P_l i(m_l) - P_r i(m_r)$$

where $m_p$ stands for the parent node which impurity remains constant for different features, $m_l$ stands for the left child node, $P_l$ stands for the probability for going to the left child node from the parent and $m_r$ and $P_r$ are equivalent just for the right side. We now want to find the feature in our input space that maximises this change. This results in the optimisation problem for each split under construction of a tree.

$$\arg\max_{f_i, i=1..M} \Delta i(m)$$

where $M$ stands for the set of unused features at node m in a given tree.

For the impurity function we will use the Gini index.

$$i_{Gini}(m) = \sum_{y_i \neq y_j} p(y_i|m)p(y_j|m)$$

Where $p(y_i|m)$ is the probability of getting decision class $y_i$ at node $m$ and $p(y_j|m)$ is the probability of getting decision class $y_j$ at node $m$.

This results in the specific optimisation problem.

$$\arg\max_{f_i, i=1..M} \big( -\sum_{y\in Y} p^2(y|m_p) + P_l \sum_{y\in Y} p^2(y|m_l) + P_r \sum_{y\in Y} p^2(y|m_r)\big)$$

The optimisation problem is solved for each split in the decision tree and the parent impurity for the root node is set to 1.

### 6.1.2 Bagging

We now introduce the first tree ensemble method called bagging. In section 6.2 in [18] it is stated that this technique builds upon bootstrapping by constructing $k$ decision trees each based on a bootstrapped subset of the training data. Each subset contains exactly the same number of samples as the training set and each sample is picked from the training set with equal probability and replacement.

The advantage of bagging over the normal decision tree is that it decreases overfitting and variance of the model meanwhile also increasing accuracy and stability of the classification.

### 6.1.3 Random Forest

As bagging, random forest is a tree ensemble method based on $k$ bootstrapped subsets of the training data, also described in section 6.4 in [18]. The bootstrapped data sets are of the same size and sampled with equal probability and replacement.

Random forest differs from bagging in the construction of each decision tree. At every split we randomly pick $m$ features without replacement and equal probability. Only from these $m$ features we find the optimal feature for the split. All $m$ features are then returned to the set of features and can potentially be picked for the next split again.

Often $m$ is chosen to be $\sqrt{|P|}$ where $|P|$ is the cardinality of the full set of features but when fitting a random forest it should be considered a hyperparameter and be fitted as well as the number of trees in the random forest.

### 6.1.4 Gradient Boost

Boosting is a general learning idea that can be used with all learners, but in our thesis we will focus on decision trees. As random forest and bagging, boosting relies on the knowledge of the crowd by combining a lot of weak learners[4] b(x).

One huge difference is, instead of training each learner individually on bootstrapped data, boosting trains each learner on the basis of the combined error of all the previous learners.

In the book [19], section 10.2 the following algorithm is stated.

**Algorithm 1** Forward Stage-wise Additive Modeling

```
initialize $f_0(x) = 0$
for $m = 1 : M$ do
    $(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$
    $f_m(x) = f_{m-1}(x) + \beta_m b(x, \gamma_m)$
end for
```

In the algorithm $\gamma$ parameterise how the decision tree is constructed, i.e. features and how to split. The book states that boosting is a way of fitting an additive expansion of a set of weak learners. Hence, the algorithm states the essence of boosting.

If we take a closer look at the optimisation problem in algorithm 1 and consider it for decision trees it takes the form.

$$\Theta_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m))$$

Where $\Theta_m = \{R_{jm}, \gamma_{jm}\}\ for\ j = 1..K$, meaning that $\gamma_{jm}$ is the construction of each tree and $R_{jm}$ is the terminal regions of the decision trees, i.e. the elements each leaf describes.

This optimisation problem is almost in every setting infeasible to solve if we are not given the optimal regions of $R_{jm}$. We can however resort to numerical optimisation to approximate the

[4]A learner that is just better than flipping a coin

solution by solving the following.

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i))$$
$$\hat{f} = \arg\min_{f} L(f)$$

The tool we will utilise is the gradient descent method.

$$f_m = f_{m-1} - \rho_m g_m$$

Where $g_m$ is the gradient of the loss function for $f_{m-1}$.

$$g_m = \left[\frac{\partial L(y, f(x))}{\partial f(x)}\right]_{f=f_{m-1}}$$

and $\rho_m$ is the step length

$$\rho_m = \arg\min_{\rho} L(f_{m-1} - \rho g_m)$$

This leads to the generic form of the gradient boosting algorithm stated in section 10.10.3 of [19], which is a combination of the forward stage-wise additive modeling algorithm and the gradient descent idea. We will use multi classification, and therefore the presented algorithm will have the softmax as loss function, which is described in greater detail under Feed-Forward Neural Networks.

$$\begin{aligned} L(y_i, f_{k,m}(x_i)) &= -\sum_{j=1}^{K} I(y_i = j)\log(p(y_i = j|x_i)) \\ &= -\sum_{j=1}^{K} I(y_i = j)\log(softmax(f_{k,m}(x_i)) \\ &= -\sum_{j=1}^{K} I(y_i = j)\log(\frac{e^{f_j(x_i)}}{\sum_{l=1}^{K} e^{f_l(x_i)}}) \end{aligned}$$

**Algorithm 2** Gradient Tree Boosting for Classification

```
initialize $f_{k,0}(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
for $m = 1 : M$ do
  for $k = 1 : K$ do
    for $i = 1 : N$ do
      $\mathrm{r}_{i,k,m} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{k,m-1}}$
    end for
    Fit a decision tree to all the $r_{i,k,m}$ giving the terminal regions $R_{j,k,m} = 1, 2, , J_{k,m}$
    for $j = 1 : J_{k,m}$ do
      $\gamma_{j,k,m} = \arg\min_\gamma \sum_{x_i \in R_{j,k,m}} L(y_i, f_{k,m-1}(x_i) + \gamma)$
    end for
    $f_{k,m} = f_{k,m-1}(x) + \nu \sum_{j=1}^{J_{k,m}} \gamma_{j,k,m} I(x \in R_{j,k,m})$
  end for
end for
Return $\hat{f}(x) = f_{k,M}(x)$
```

The part of the gradient boosting algorithm that takes the most time is by far fitting all the decision trees and due to its sequential nature we can not parallelise the process as in bagging and random forest. Instead we will use the LightGBM [20] implementation that uses some different techniques to speed up the training process.

There are two ways we can find the split points in the decision trees. The first is known as the pre-sorted algorithm which goes through all the possible splits on sorted data. This is the classic way which the generic form of the gradient boost algorithm implements, and it is very time and memory consuming.

The second method is called the histogram-based algorithm, which is used in the LightGBM implementation. This algorithm gives up a bit of precision but in return we get a huge time and memory gain. It does this by splitting up the data into bins or quantiles based on the weights $r_{i,k,m}$. This way badly classified samples are attached more importance compared to well classified samples. The histogram algorithm uses approximately the same time sorting the data into histograms as the pre-sorted algorithm uses to sort the data, but then the pre-sorted needs to go through all the data, and the histogram-based only needs to consider the quantiles.

Another technique the LightGBM introduces is called Gradient-based One Side Sampling(GOSS). It reduces the data size by under-sampling samples with a low gradient because they are already well classified and do not need more attention.

One important thing to mention is that just discarding all the well trained samples is not desirable. It will alter the data distribution leading to a bad over all learner.

One last technique LightGBM implements is Exclusive feature bundling. It pools features that are mutually exclusive together. This is not that useful for our data because after feature selection the number of features is not that huge an issue for us.

## 6.2 Feed-Forward Neural Networks

The architecture of the human brain has for a long time served as inspiration for a way to automatise learning tasks. Feed-Forward Neural Network tries to map the input layer to the output layer through a structure of nodes in between. The intermediate nodes are commonly denoted as hidden layers. A Feed-Forward Neural Network consisting of an input layer, $k$ hidden layers, and an output layer can be seen as a $(k+2)$-partite graph. In the following sections we present a general introduction to important considerations when designing a Neural Network-classifier.

### 6.2.1 Architecture

Let the input layer be given by an input vector $\boldsymbol{x} = (x_1, x_2, ..., x_{n-1}, 1)$ with dimension $(1, n)$, where each element $x_i$ for $i \in \{1, ..., n-1\}$ in the vector corresponds to a sample feature. The constant 1 is added to include a bias term.

Let $\boldsymbol{W^{(l)}}$ denote a *weight matrix* of dimension $(m \times n)$ corresponding to the transformation between layer $(l-1)$ and $l$. It can be seen as a mapping $M^{(l)} : \mathbb{R}^n \to \mathbb{R}^m$ in which n is the number of entries in the input vector $\boldsymbol{x}$, and m is the number of entries in the output vector $\boldsymbol{y}$. Let the specific mapping be given by $M^{(l)}(\boldsymbol{x}) = \boldsymbol{W^{(l)}} \cdot \boldsymbol{x^T} = \boldsymbol{y}$.

A Feed-Forward Neural Network consists of stacking multiple layers together. Introducing the concept of an *activation function* will allow the network to mimic highly non-linear functions. Let a non-linear activation function be denoted by $\phi$ and let $\boldsymbol{z} = \phi(\boldsymbol{y})$ where $\phi$ is applied to each entry of the vector $\boldsymbol{y}$.

Combining the concepts introduced so far leads to the following definitions of each entry:

$$\mathrm{y}_i^{(l)} = (M^{(l)}(\boldsymbol{x}))_i = (\boldsymbol{W^{(l)}} \cdot \boldsymbol{x^T})_i = \textstyle\sum_{j=1}^{n} x_j \cdot w_{i,j}^{l}$$

$$\mathrm{z}_i^{(l)} = \phi(M^{(l)}(\boldsymbol{x}))_i = \phi(\boldsymbol{W^{(l)}} \cdot \boldsymbol{x^T})_i = \phi(\textstyle\sum_{j=1}^{n} x_j \cdot w_{i,j}^{l})$$

To demonstrate let a Feed-Forward Neural Network consist of an input layer, two hidden layers, and an output layer. In the output layer $l = 3$, which corresponds to the vector $\boldsymbol{z^{(3)}}$.

$$\boldsymbol{z^{(3)}} = \phi(M^{(3)}(\phi(M^{(2)}(\phi(M^{(1)}(\boldsymbol{x}))))))$$

The composite function is named the *hypothesis*, and it is usually denoted by $h_{\boldsymbol{\Theta}}(\boldsymbol{x})$. Here $\boldsymbol{\Theta}$ refers to all the weight matrices $\boldsymbol{W^{(l)}}$.

### 6.2.2 Cost Functions

In order to train the weight of the neural network we need a way to quantify the error made in the classification task. This is done by introducing a cost function. A cost function is essentially an average of the losses associated with each prediction. After choosing such a function the goal is to minimise it.

$$\arg\min_{\boldsymbol{\Theta}} J(\boldsymbol{\Theta}) = \frac{1}{m}\sum_{i=1}^{m} \mathbb{L}_i(h_{\boldsymbol{\Theta}}(\boldsymbol{x})_i, \boldsymbol{y}_i)$$

The specifics of the loss function L is of high importance to the quality of the trained classifier [21]. The loss function is critical to handle non-convexity of the space, which in turn ensures non-ambiguity with regards to the optimal local minimum. Furthermore the specifics determine how fast convergence towards optimum will occur, since the gradient with respect to the cost function should be controlled in a way such that we move towards the optimal solution in the best possible way. One of the most common loss functions used is *log loss* [21].

$$\mathbb{L}(h_{\boldsymbol{\Theta}}(\boldsymbol{x})_i, \boldsymbol{y}_i) = -\sum_{j}[y_j \log p(o_j)]$$

In the above $y_j$ refers to the j'th element of true class labels encoded using one-hot-encoding. For the predicted labels $o_j$ refers to the j'th element.

### 6.2.3 Activation Functions

In the introduction the *activation function* $\phi$ was introduced. Non-linear activation functions makes it possible for the models to capture non-linear behavior. Many activation functions exist, and they all have pros and cons. A well-known problem is vanishing gradient in which the gradient gets too small because of multiplication of small values during back-propagation [22]. This leads to small updates and slow convergence towards the minimum for the loss function. To avoid the above described issue the following activation functions are proposed in this project.

Table 1: Activation Functions and their derivatives

| Name | Formula | Derivative |
|---|---|---|
| Rectified Linear Unit [ReLU] | $\phi_i(\boldsymbol{y}) = \max(0, y_i)$ | $\frac{\partial \phi_i(\boldsymbol{y})}{\partial y_i} = \begin{cases} 1 & y_i > 0 \\ 0 & y_i \leq 0 \end{cases}$ |
| Softmax | $\phi_i(\boldsymbol{y}) = \frac{e^{y_i}}{\sum_j^m e^{y_j}}$ | $\frac{\partial \phi_i(\boldsymbol{y})}{\partial y_i} = \begin{cases} \phi_i(\boldsymbol{y})(1-\phi_i(\boldsymbol{y})) & i = j \\ -\phi_i(\boldsymbol{y})\phi_j(\boldsymbol{y}) & i \neq j \end{cases}$ |

Each activation function posses different properties. ReLU is often used in the hidden layers because it posses linearity for positive values. This makes it robust to the vanishing gradient problem. Notice how it transforms each element of the output vector $\boldsymbol{y}$. The nature of the SoftMax function resembles a valid probability distribution, which makes it ideal for the output layer in multi-class classification tasks [22]. To do this it takes the entire output vector $\boldsymbol{y}$ into account.

### 6.2.4 First-Order Stochastic Optimization: ADAM

The task of training the network means to update the weight matrices in such a way that the loss function is minimised. One way to do this is by using the gradient. Instead of running through every training example for each iteration, the algorithm randomly selects training examples to represent an estimated gradient for the loss function. The number of training examples used to approximate the gradient is known as the *batch size* and it gives the algorithm a stochastic nature. Let $\boldsymbol{\Theta}$ denote a vector of weight matrices $\boldsymbol{W}^{(l)}$.

**Algorithm 3** ADAM

```
Require: $\alpha$: Stepsize
Require: $\epsilon$: Small number to avoid division by zero
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\Theta)$: Stochastic objective function with weight matrices $\Theta$
Require: $\Theta_0$: Initial weight matrices.
$m_0 \leftarrow 0$ (Initialise first moment vector)
$v_0 \leftarrow 0$ (Initialise second moment vector)
$t \leftarrow 0$ (Initialise timestep)
while $\Theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\Theta f_t(\Theta_{t-1})$
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$
    $\hat{m}_t \leftarrow m_t/(1 - \beta_1^t)$
    $\hat{v}_t \leftarrow v_t/(1 - \beta_2^t)$
    $\Theta_t \leftarrow \Theta_{t-1} - \alpha \cdot \hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon)$
end while
return $\Theta_t$
```

The ADAM algorithm, which is taken from the original paper [23], was presented at International Conference for Learning Representation in 2015. It combines the advantages of using AdaGrad and RMSProp, which are two other popular algorithms for first-order optimisation. AdaGrad works well for sparse gradients, and RMSProp works well for on-line and non-stationary settings.

The way it works is somewhat analogous to momentum in a physical setting. By incorporating information about previous gradients it allows to by-pass local minima in which the gradient is 0. Furthermore, it adjusts as a result of how much the gradient varied in the past.

The parameter $t$ denotes the iteration number, which simply means how many times the while-loop has been executed. The gradient $\nabla_\Theta f_t(\Theta_{t-1})$ is the matrix of partial derivatives with respect to the weight matrix $\Theta$. The first moment vector $m_t$ is the expected value of the gradient at iteration $t$, and the second moment vector $v_t$ is the expected value of the gradient element-wise squared. It can be shown by expanding the recursive formula [23]. After the expansion we arrive at the following expression.

$$\begin{aligned} m_t &= \mathbb{E}[g_t] \cdot (1-\beta_1^t) + \psi \\ v_t &= \mathbb{E}[g_t^2] \cdot (1-\beta_2^t) + \psi \end{aligned}$$

The reason they can be interpreted as expected values is that the algorithm samples random batches. Inserting the new expression into respectively $\hat{m_t}$ and $\hat{v_t}$ gives the following.

$$\begin{aligned} \hat{m_t} &= \tfrac{\mathbb{E}[g_t]\cdot(1-\beta_1^t)+\psi}{(1-\beta_1^t)} \approx \mathbb{E}[g_t] \\ \hat{v_t} &= \tfrac{\mathbb{E}[g_t^2]\cdot(1-\beta_2^t)+\psi}{(1-\beta_2^t)} \approx \mathbb{E}[g_t^2] \end{aligned}$$

Combining the terms above gives a clear way to interpret the updates made by the algorithm.

$$\Theta_t \leftarrow \Theta_{t-1} - \alpha \cdot \hat{m_t}/(\sqrt{\hat{v_t}} + \epsilon) \approx \Theta_{t-1} - \alpha \cdot \mathbb{E}[g_t]/(\sqrt{\mathbb{E}[g_t^2]} + \epsilon)$$

Whenever the ratio between the uncentered variance $\mathbb{E}[g_t^2]$ and the mean $\mathbb{E}[g_t]$ is large, then we have good confidence in the update direction. This means that a larger step size is appropriate. Likewise, when the uncertainty is high, it will be conservative with the step size.

The algorithm has 4 hyperparameters $\epsilon, \beta_1, \beta_2$, and $\alpha$. The authors achieved good results with default settings $\epsilon = 10^{-8}, \beta_1 = 0.9, \beta_2 = 0.999$, and $\alpha = 0.001$ [23].

## 6.3 Summary

We have now investigated the theory behind the random forest model, gradient boosting and neural networks. The gradient boosting and the neural network models where chosen due to their wide spread use and success in different application. Especially their success in Kaggle competitions was a deciding factor, when we searched for models to include in the thesis.

The random forest model was included to have another decision tree based model to compare the gradient boosting machine to. It has also been around for a long time and was a natural predecessor to the gradient boosting idea.

In the proceeding sections theory about implementation of the mentioned models will be introduced and their performance assessed on the NSL-KDD99 data set together with earlier mentioned preprocessing techniques.

# 7 Training Theory

The main objective of training a classifier is to fit according to the training data without lowering its ability to generalise on unseen data. The latter is important because a model without the ability to classify unseen data is useless. When a model has a really good ability to classify the training sample, but has a poor ability to classify unseen data, we call it overfitted. Likewise, when a model has a poor ability to determine the underlying structure of the training data, it is underfitted.

In this section we wish to elaborate on the ways we will be training our classifiers. These techniques are used to maintain the ability to generalise on unseen data with highest possible performance.

## 7.1 Tuning Hyperparameters

Hyperparameters are parameters that must be set before training a classifier. It can be parameters chosen for the specific optimisation algorithm, but it can also be for the structure of the models. Choosing hyperparameters is like making assumptions about the data, and therefore we need to provide justification for these assumptions.

### 7.1.1 Cross-Validation

A way to test the chosen model structure and optimisation algorithm is by cross-validation. This serves to estimate how well the model performs in a general setting. A common technique is K-fold Cross Validation.

**Algorithm 4** K-fold Cross Validation

```
Let X be the training data consisting of n samples
Let $x_i$ where $i \in \{1, .., k\}$ be $\frac{n}{k}$ samples from X such that $x_i \cap x_j =$ for $i \neq j$.
Let $s = 0$
for $j = 1$ to $k$ do
    Train the model on $x_i$ for $i \in \{1, .., k\}/j$.
    Compute performance metric $PM_j$ on unseen $x_j$.
    $s = s + PM_j$
end for
Return $\frac{s}{k}$
```

If the model with a certain structure performs better on unseen data than a model with a different structure, then it is assumed that it is the better choice. As the model has to be fitted in each iteration, it is a computationally expensive approach. In the community, most people use 3, 5, or 10 folds.

Repeating cross-validation might seem tempting in order to use statistical inference and obtain useful information about the entire domain. However, an estimate for statistics in the population consisting of all k-folds is of little interest. In an article on this subject Gitte Vanwinckelen and Hendrik Blockeel argue that confidence intervals based on repeated cross-validation are often misleading [24]. Hence, we have deliberately avoided such efforts.

### 7.1.2 Grid-Search

When searching for specific settings for algorithms and model structure it is useful to approach the task systematically. By estimating a range of values for each hyperparameter one simply tries them all.

**Algorithm 5** Grid-Search

```
Let H denote the n-dimension hyperparameter space consisting of p elements.
Let R be an initially empty array of p elements.
for $j = 1$ to $p$ do
    Use K-fold Cross Validation on model with hyperparameters $H_j$. Save result in $R[j]$.
end for
Return the index of the maximum value in R and use it to find the hyperparameters used.
```

For complex structures, which often require a lot of time to fit, this can take a very long time. Therefore, it is important to estimate reasonable ranges for the parameters by using knowledge from the specific setting or by searching through parameters used in similar settings.

### 7.1.3 Random Search

Using Grid-Search can be extremely time consuming. An modified version, known as Random Search, can be used to randomly sample configurations of hyperparameters from the grid. By considering each sampling as independent trials, one can derive a formula deciding how many samples to include.

We wish to find a configuration $c$ among the $t$ percent best of the entire search space. Drawing such a configuration has probability of exactly $P[c \in \text{top } t \text{ percent}] = t$. As a result $P[c \notin \text{top } t \text{ percent}] = 1 - t$. We wish to find the result with a confidence $k$ after drawing $n$ samples.

$$\begin{aligned}
1 - (1-t)^n &> k \\
1 - k &> (1-t)^n \\
\log(1-k) &> \log(1-t) \cdot n \\
\frac{\log(1-k)}{1-t} &< n
\end{aligned}$$

In the experiments conducted we wished to find a hyperparameter configuration in top 5 percent with 95 percent confidence. By using the above formula, this resulted in n = 59, when rounded to nearest integer.

## 7.2 Ensembles

Ensembles are wide spread in machine learning and is a general expression for a combination of models. We have already introduced two models which builds upon the ensemble idea consisting of the gradient boosting model and the random forest model. We use ensembles as a mean for reducing variance of the model. If we have 10 models and each model is trained upon the same data but each having some variance, we can use the fact that when having more samples the population mean will converge towards the true mean.

### 7.2.1 Variance Reduction in Neural Networks

Ensemble methods often consist of different models or the same model but trained on subsets of the same data set. However, in section 7.11 of [25] they describe that even ensembles of the same neural network trained on the same data can benefit from an ensemble of these. The reason for this is that random initialisation of weights and different outcomes of non-deterministic implementations will lead to different models leading to a generic high variance of neural networks. We therefore train 6 neural networks and use them as an ensemble in our thesis. The reason for picking 6 comes from experimental results where the variance was found to be more or less stable for ensembles consisting of 6 or more neural networks.

### 7.2.2 Ensemble of Different Models

All ensembles introduced until now have been ensembles of the same base model, but we can also create ensembles of different models. If we see different base learners performing well in different areas we can often combine them to obtain better results. If we take a look at table 2, which shows the confusion matrices of the ensemble of neural networks and the gradient boosting machine, we see that they make different errors. Especially the neural networks predicts R2L well but lack on probe while the gradient boosting machine is the opposite story. We can therefore by combining them hopefully improve the ability to generalise on unseen data.

Table 2: Confusion matrix for neural networks and gradient boosting machine trained on CFS-SMOTE modified data presented in section 9.6

| | **Neural Networks** | | | | | **Gradient Boosting** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **DoS** | **Normal** | **Probe** | **R2L** | **U2R** | **DoS** | **Normal** | **Probe** | **R2L** | **U2R** |
| **DoS** | **5816** | 1006 | 146 | 492 | 0 | **5737** | 1698 | 23 | 0 | 2 |
| **Normal** | 55 | **9112** | 193 | 272 | 79 | 67 | **9401** | 228 | 7 | 8 |
| **Probe** | 221 | 639 | **1480** | 63 | 18 | 165 | 170 | **2080** | 4 | 2 |
| **R2L** | 10 | 1765 | 14 | **927** | 169 | 0 | 2675 | 27 | **162** | 21 |
| **U2R** | 0 | 5 | 0 | 21 | **41** | 0 | 25 | 0 | 2 | **40** |

# 8 Performance Metrics

In order to evaluate the performance of different classifiers we need a way of quantifying their performance. However, as one is often faced with imbalanced data sets we must carefully select the metrics so they are not misleading [26]. For the following a few important definitions are needed. These are usually used for binary classification, but the following is an attempt to generalise the terms.

**True positive [TP]:** The classifier correctly predicts the presence of class label A
**True negative [TN]:** The classifier correctly predicts the absence of class label A
**False positive [FP]:** The classifier predicts the presence of class label A, when it is absent
**False negative [FN]:** The classifier predicts the absence of class label A, when it is present

In the following section we will present some of the metrics and visualisations we use for evaluating our classifiers. These are divided into two categories: Namely the ones sensitive to imbalanced data and the ones that are insensitive. The latter will be referred to as *robust metrics*. Alaa Tharwat [26] describes an easy way to distinguish between these. Consider the confusion matrix in the section below. Any metric calculated from one row only will be insensitive to imbalanced data, because looking at one row will reveal nothing about the distribution between labels.

## 8.1 Robust Metrics

### 8.1.1 Confusion Matrix

A confusion matrix, which is also known as a contingency table, has predicted classes and true classes placed in respectively columns and rows.

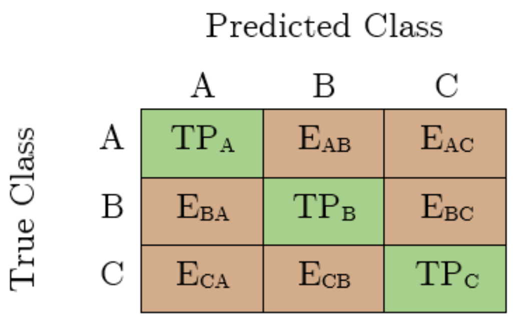

Figure 11: Example of a confusion matrix

A good classifier will have most of its values placed in the diagonal, which corresponds to *true positives*. A confusion matrix is not sensitive to imbalanced data, since it is a raw presentation of predictions and true observations [26]. Introducing color gradients may however lead to a false impression which is why it is necessary to include the numbers or percentages.

### 8.1.2 Recall [R]

It is a ratio between the number of correctly classified positive samples and the total number of positive samples.

$$\mathrm{R} = \frac{TP}{TP+FN}$$

It can be interpreted as the classifiers ability to correctly detect the presence of a class label. It is used interchangeably with the terms *true positive rate*, *sensitivity*, and *hit rate* [27].

### 8.1.3 False Positive Rate [FPR] and False Negative Rate [FNR]

These are metrics of high importance to intrusion detection systems. The definitions are as follows.

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

$$\mathrm{FNR} = \frac{FN}{FN+TP}$$

FPR can be seen as *false alarm rate* [26]. It is desirable to decrease this as much as possible to ensure trust in the detection systems. Furthermore reacting on such alarms have operational costs, which are desirable to minimise.

FNR can be seen as *miss rate* [26]. For obvious reasons it is desirable to be able to detect the attacks for which the classifier was designed. Hence, the objective is to minimise FNR.

## 8.2 Sensitive Metrics

### 8.2.1 Accuracy [A]

A natural way of quantifying the performance is to look at the number of correct predictions. It is defined as follows.

$$\mathrm{A} = \frac{TP+TN}{TP+TN+FP+FN}$$

It can be calculated from the confusion matrix by finding the ration between the sum of the diagonal and the sum of the entire matrix.

There are several problems related to using accuracy as a measure of performance. The method risks giving a false impression for imbalanced data. Consider a binary classifier used to classify 100 examples. 95 of the examples are positive cases and 5 are negative. By predicting positive in each example the classifier would reach an accuracy of 95%. This does however not say anything useful about its ability to model the situation.

Another problem is ambiguity. It is not guaranteed that two classifiers with the same accuracy makes identical decisions [26]. Hence, the accuracy alone does not provide good basis for comparison.

### 8.2.2 Precision [P]

It is the ratio of correctly predicted positives to the total number of predicted positives.

$$\mathrm{P} = \frac{TP}{TP+FP}$$

It can be interpreted as the extend to which a classifier can be trusted when it predicts the presence of a class label.

### 8.2.3 F-measure [F]

It is the harmonic mean of precision and recall. Since it depends on precision, it is sensitive to changes in the class distribution [26]. It is defined as follows:

$$\mathrm{F} = \frac{2 \cdot P \cdot R}{P+R}$$

A high F-measure (i.e. close to 1) can be seen as an indicator of good performance.

## 8.3 Receiver Operating Characteristics [ROC] and Area Under the Curve [AUC]

The ROC-curve is a 2-dimensional graph consisting of true positive rate on the y-axis and false positive rate on the x-axis. Binary classifiers produce a score corresponding to their confidence in their predictions. Usually these are strict probabilities meaning they lie in the interval $[0, 1]$. By varying the value of the threshold distinguishing positive from negative samples we can generate points (**TPR**, **FNR**) on the ROC-curve [26].

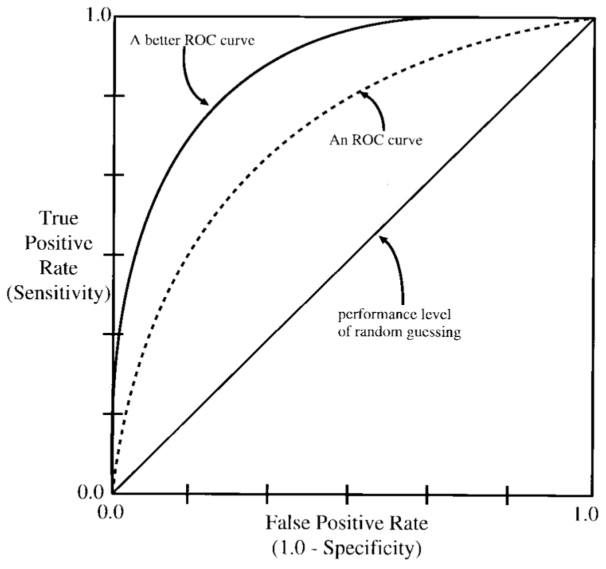

Figure 12: Overview of ROC-curve from article by Kevin Woods and Kevin Bowyer [3]

The ROC-curves come with several uses. Firstly, they illustrate how deciding on a threshold for a classifier is a trade-off. There is no obvious answer to which point on a curve that is best, since it depends entirely on the context for which it is used. In the context of IDS, it is desirable to keep the FPR down, since taking action on an alarm comes with some cost. Furthermore, too many false alarms will cause a distrust in the system. A low FPR will however result in a low TPR, which is also undesirable, since the IDS will detect fewer attacks.

Secondly, it illustrates whether a classifier performs better than random guessing as illustrated by the straight line in the diagonal.

Lastly, they can be used to compare the performance of classifiers. Regardless of the trade-off a classifier that lies to the left of another is generally considered better. This is because the ratio between TPR and FNR will be higher for all points on the curve [26].

Sometimes it is desirable to compare single scalar values for different classifiers. Calculating the area under the curve, AUC, gives such a value. The AUC is the probability that a classifier will assign a higher score to a random positive sample than a random negative sample [28]. Notice in Figure 12 how the AUC for the random guessing with be 0.5. Hence, a proper classifier should always be above 0.5.

# 9 Comparative Analysis on Trained Models

Based on the theoretical considerations outlined in the thesis the empirical work followed the flow in Figure 13.

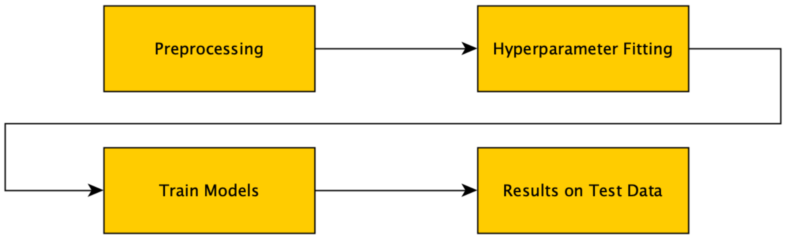

Figure 13: Overview of workflow

Before doing any preprocessing, we grouped the attack types into 5 major classes: Normal, DoS, Probe, R2L, and U2R.

Preprocessing is a wide term for any manipulation done to the data prior to creating models. In our case, it involves 3 operations: Imbalance Correction, Feature Selection, and One-Hot-Encoding on nominal features. Commonly, this part also includes normalisation of data. Since we did not gain any significant advantage from doing so in the preliminary trials, we deliberately decided not to do this.

The data sets will be, depending on types of preprocessing operations that has been made on them, referred to in the following way.

Table 3: Names for data sets

| **Imbalance-Corrected** | **Feature-Selection Method** | **Data Set Name** |
|---|---|---|
| None | None | Raw data set |
| SMOTE | None | SMOTE data set |
| None | CFS | CFS data set |
| None | FCBF | FCBF data set |
| None | Consistency | Consistency data set |
| SMOTE | CFS | CFS SMOTE data set |

The hyperparameter fitting was carried out by choosing a parameter space on which we performed a search technique to find the optimal set of features for a given model. We have used random search and grid search to find the optimal parameters for the earlier discussed models.

The found optimal hyperparameters were then used in the training of the models and then evaluated on the test data.

The original NSL-KDD99 data sets, the preprocessed data sets, the trained models, and the code to generate these are attached to the thesis. Appendix E contains the content of the textfile "readme.txt", which comes with the zipped folder.

In the following sections we will elaborate on the individual results.

## 9.1 Results from Imbalance Handling and Feature Selection

The imbalance correction was done using the SMOTE algorithm. We obtained an almost perfectly balanced data set with 25194 instances of probe, and 25195 instances of respectively DoS, normal, U2R, and R2L.

The other tool we applied in the preprocessing was feature selection. We used the feature selection methods CFS, FCBF, and a consistency based method. The features selected by each method are shown in Table 4.

Table 4: Selected features for each method

| | Selected Features |
|---|---|
| CFS: | x3, x4, x5, x6, x12, x25, x29, x30, x37, x39 |
| FCBF: | x5, x25 |
| Consistency: | x1, x3, x5, x6, x12, x23, x32, x33, x35, x37, x38, x39, x40 |

The effect of the imbalance correction and each of the feature selection methods will be investigated in the following sections.

## 9.2 Specifications of Models Used

### 9.2.1 Gradient Boosting Machine

Four different hyperparameters were fitted for the gradient boosting machine model. These where number of trees, the maximum depth of each tree, the maximum number of leaves, and minimum data points each leaf in a tree was allowed to contain. Because grid search is a very heavy search method we initially did a manual search on single parameter combinations in a very large range. When a feasible range was found the grid search was applied for each data set to find the optimal set of hyperparameters.

The found range for the grid search was:

trees $= \{465, 475, 490\}$
max depth $= \{23, 24, 25\}$
max leafs $= \{165, 170, 175\}$
min samples $= \{4, 5, 6\}$

The learning rate was set to 0.005 because it did not result in divergence. Therefore we saw no reason for trying lower values because it would only result in slower training.

### 9.2.2 Random Forest

The random forest has two hyperparameters which were fitted following the same approach as in the gradient boosting machine.

The final range to apply grid search over was found to be:

trees $= \{300, 350, 400\}$
split features $= \{22, 24, 26\}$

### 9.2.3 Neural Network

For neural networks we used random search to find suitable hyperparameters for the networks. We tried 59 different configurations, which corresponds to a 95 percent certainty that at least one configuration is within top 5 percent best result in the chosen hyperparameter space. We chose the following space:

learning rates $= \{4 \cdot 10^{-3}, 1 \cdot 10^{-3}, 7 \cdot 10^{-4}, 4 \cdot 10^{-4}, 1 \cdot 10^{-4}, 7 \cdot 10^{-5}, 4 \cdot 10^{-5}, 1 \cdot 10^{-5}\}$
hidden layers $= \{2, 4, 6, 8, 10, 12, 14\}$
hidden units $= \{50, 75, 100, 125, 150, 175, 200, 225, 250, 275, 300, 325, 350\}$

The tuning is based on the assumption that the optimal configurations can be found in the chosen space. The learning rate was considered around the default value of the Adam optimiser, since the default value was chosen by the developers to be versatile. For the hidden layers and the number of weights, computational feasible sizes were chosen.

At first glace, the results seemed to indicate that deeper networks with a large number of weights would lead to better performance, since they performed the best on the validation set taken from the training data. When tested on the test data we saw a huge bias towards the U2R and normal class. Specifically we present the confusion matrix of a network with 6 hidden layers with 275 units each in Table 6 trained on the CFS SMOTE data set, but the pattern was similar for all the data sets.

This lead us to think that the large models overfitted the training data. To tackle this problem we restricted us to only consider shallow networks with only two layers. This is a form of manual regularisation and this could also have been implemented automatically in the selection procedure.

The best performing network that satisfied this had 2 hidden layers and 200 units in each. It can be seen from Table 6 that the classification is much more uniform than for the one with 6 hidden layers and 275 units in each.

Table 5: Confusion matrix for neural networks trained on CFS SMOTE data set and tested on the test data.

| | **2 hidden layers, 200 units per layer** | | | | | **6 hidden layers, 275 units per layer** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **DoS** | **Normal** | **Probe** | **R2L** | **U2R** | **DoS** | **Normal** | **Probe** | **R2L** | **U2R** |
| **DoS** | **5816** | 1006 | 146 | 492 | 0 | **5369** | 647 | 76 | 25 | 1343 |
| **Normal** | 55 | **9112** | 193 | 272 | 79 | 36 | **8643** | 285 | 31 | 716 |
| **Probe** | 221 | 639 | **1480** | 63 | 18 | 153 | 250 | **1997** | 12 | 9 |
| **R2L** | 10 | 1765 | 14 | **927** | 169 | 3 | 2025 | 20 | **209** | 628 |
| **U2R** | 0 | 5 | 0 | 21 | **41** | 0 | 11 | 0 | 0 | **56** |

We achieved good results for the model with 2 hidden layers and 200 units for all the data sets. Therefore we decided to settle on this design. We used ReLU as the activation function in the hidden layers and softmax for the output layer due to the multi classification problem.

## 9.3 Specification of Ensemble

The ratio between the gradient boosting machine and the neural networks can be thought of as a "hyper-hyper"-parameter. Therefore it should also be fitted before testing the model on an unseen test set.

It is a problem however, that the training data is already used to train the models that constitute the ensemble. We can therefore not use this data to get an informative answer regarding the best ratio between the models in the ensemble.

We therefore tried to fit the models on the Raw data set and then fit the ensemble on the SMOTE data set. The idea was to mimic new data by introducing to all the synthetic points. This did not work as intended. Table 6 shows that the gradient boosting machine is either correct or normal. This could indicate that the gradient boosting machine is overfit on the data.

Another thing that spoke against this approach was that the ensemble of the neural network and the gradient boosting machine always had a lower accuracy than the pure neural network and a lower F1 score than the pure gradient boosting machine.

We believe one of the reasons for these problems was that some of the training data was already known to the models so these was trivially classified. There we concluded that fitting on the synthesized training data was infeasible. We therefore decided to split up the test data in a training part and a test part. This has the risk of not generalising to other attack distributions, and some of the idea behind testing on another distribution than the training data disappears. With that being said, we still see that the neural network exhibits a better performance in the R2L class in both the confusion matrix in Table 6 and 7. This provides us confidence in the belief that the ensemble would be able to generalise to other distributions.

Table 6: Confusion matrix for neural networks and gradient boosting machine trained on the Raw data set and tested on the SMOTE data set

| | **Neural Networks** | | | | | **Gradient Boosting** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **DoS** | **Normal** | **Probe** | **R2L** | **U2R** | **DoS** | **Normal** | **Probe** | **R2L** | **U2R** |
| **DoS** | **24415** | 547 | 233 | 0 | 0 | **23621** | 1574 | 0 | 0 | 0 |
| **Normal** | 152 | **24765** | 264 | 14 | 0 | 0 | **25192** | 3 | 0 | 0 |
| **Probe** | 152 | 1739 | **23280** | 23 | 0 | 0 | 9681 | **15513** | 0 | 0 |
| **R2L** | 3 | 18737 | 847 | **5608** | 0 | 5 | 21943 | 0 | **3240** | 7 |
| **U2R** | 20 | 24372 | 788 | 15 | **0** | 0 | 17323 | 0 | 0 | **7872** |

Table 7: Confusion matrix for neural networks and gradient boosting machine trained on CFS SMOTE data set and tested on the test data

| | **Neural Networks** | | | | | **Gradient Boosting** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **DoS** | **Normal** | **Probe** | **R2L** | **U2R** | **DoS** | **Normal** | **Probe** | **R2L** | **U2R** |
| **DoS** | **5816** | 1006 | 146 | 492 | 0 | **5737** | 1698 | 23 | 0 | 2 |
| **Normal** | 55 | **9112** | 193 | 272 | 79 | 67 | **9401** | 228 | 7 | 8 |
| **Probe** | 221 | 639 | **1480** | 63 | 18 | 165 | 170 | **2080** | 4 | 2 |
| **R2L** | 10 | 1765 | 14 | **927** | 169 | 0 | 2675 | 27 | **162** | 21 |
| **U2R** | 0 | 5 | 0 | 21 | **41** | 0 | 25 | 0 | 2 | **40** |

The test data was split into a training set consisting of 15780 samples and test set consisting of the remaining 6764 samples. We then fitted all ensembles on the new training data and the found ratios can be seen in Table 8.

Table 8: Specifications for ensembles of neural networks and gradient boosting machines

| **Data** | **No. Neural Networks** | **Gradient Boosting Ratio** | **Neural Network Ratio** |
|---|---|---|---|
| Raw | 5 | 68% | 32% |
| Smote | 4 | 75% | 25% |
| CFS | 6 | 71% | 29% |
| FCBF | 5 | 69% | 31% |
| Consistency | 4 | 68% | 32% |
| Smote and CFS | 2 | 75% | 25% |

## 9.4 Effect of Imbalance Correction

First we compared the models trained on the Raw data set and SMOTE data set.

Table 9 and 10 show that the accuracy is not that different, but the F1-score is significantly higher for the corrected data set as seen in Table 10. The reason for similar accuracy between the corrected and the skewed training data is that accuracy is not a robust metric. Hence, being good at DoS and normal is sufficient to achieve a high accuracy since the test data is also imbalanced. On the other hand every class has equal weights in the F1 score as explained in section 8. Therefore one will never be able to score high in the F1 measure by only being able to classify two classes. This is the reason we see SMOTE getting a higher F1-score than the Raw data set.

Furthermore it is also noticeable that the FNR is lower for SMOTE and the reason for this is the same as for the F1-score.

Table 9: Metrics for training on Raw data set

| | Neural Networks | Random Forest | Gradient Boost | Ensemble |
|---|---|---|---|---|
| Accuracy | 0.72 | 0.70 | 0.69 | 0.71 |
| F1-score | 0.45 | 0.45 | 0.43 | 0.44 |
| Precision | 0.54 | 0.88 | 0.48 | 0.48 |
| Recall | 0.46 | 0.45 | 0.42 | 0.44 |
| FPR | 0.09 | 0.10 | 0.11 | 0.10 |
| FNR | 0.54 | 0.55 | 0.58 | 0.56 |

Table 10: Metrics for training on SMOTE data set

| | Neural Networks | Random Forest | Gradient Boost | Ensemble |
|---|---|---|---|---|
| Accuracy | 0.75 | 0.75 | 0.78 | 0.77 |
| F1-score | 0.57 | 0.57 | 0.65 | 0.63 |
| Precision | 0.59 | 0.77 | 0.81 | 0.79 |
| Recall | 0.62 | 0.56 | 0.62 | 0.62 |
| FPR | 0.08 | 0.08 | 0.07 | 0.08 |
| FNR | 0.38 | 0.44 | 0.38 | 0.38 |

## 9.5 Effect of Feature Selection

To assess the performance of each feature selection method we compare how well they do hold up against models trained on the Raw data set.

It is apparent from Table 12 that the FCBF method which only propose 2 features is not desired. It is worse in every aspect compared to the models trained on the Raw data set.

The consistency method is better than the FCBF method but as is shown from Table 13 it still scores worse than the Raw data set in all aspects.

The last method CFS has less features than the consistency method but has better results. It also performs better than the Raw data in almost every aspect. Especially noticeable is the lower FNR indicating it gets more of the under-represented classes right than models trained on the Raw data set.

Table 11: Metrics for training on the CFS data set

| | Neural Networks | Random Forest | Gradient Boost | Ensemble |
|---|---|---|---|---|
| Accuracy | 0.73 | 0.77 | 0.75 | 0.76 |
| F1-score | 0.45 | 0.55 | 0.49 | 0.48 |
| Precision | 0.43 | 0.80 | 0.89 | 0.70 |
| Recall | 0.48 | 0.56 | 0.49 | 0.49 |
| FPR | 0.09 | 0.08 | 0.08 | 0.09 |
| FNR | 0.52 | 0.44 | 0.51 | 0.51 |

Table 12: Metrics for training on the FCBF data set

| | Neural Networks | Random Forest | Gradient Boost | Ensemble |
|---|---|---|---|---|
| Accuracy | 0.68 | 0.74 | 0.73 | 0.73 |
| F1-score | 0.34 | 0.45 | 0.43 | 0.43 |
| Precision | 0.33 | 0.56 | 0.44 | 0.45 |
| Recall | 0.37 | 0.45 | 0.44 | 0.44 |
| FPR | 0.10 | 0.09 | 0.09 | 0.09 |
| FNR | 0.63 | 0.55 | 0.56 | 0.56 |

Table 13: Metrics for training on the Consistency data set

| | Neural Networks | Random Forest | Gradient Boost | Ensemble |
|---|---|---|---|---|
| Accuracy | 0.74 | 0.74 | 0.73 | 0.74 |
| F1-score | 0.47 | 0.48 | 0.46 | 0.47 |
| Precision | 0.62 | 0.74 | 0.69 | 0.49 |
| Recall | 0.47 | 0.48 | 0.46 | 0.47 |
| FPR | 0.09 | 0.09 | 0.09 | 0.09 |
| FNR | 0.53 | 0.52 | 0.54 | 0.53 |

## 9.6 Effect of Imbalance Correction Combined with Feature Selection

Because CFS was the only method that performed better than raw data, we decided to only combine this method with SMOTE. We first performed SMOTE on the Raw data set and then performed CFS.

Compared to the features in the CFS data set as seen in Table 4, this resulted in discarding the features x25 and x29, and in adding the features x1, x14, x17, x23, and x33. Table 14 shows that it resulted in superior performance compared to all the other data sets. Especially the ensemble of the gradient boosting machine and the neural networks stands out with an accuracy of 81% and a F1 score of 68%. This indicates that it classifies many of the samples, which belong to the minority classes, correctly.

Table 14: Metrics for training on CFS SMOTE data set

| | Neural Networks | Random Forest | Gradient Boost | Ensemble |
|---|---|---|---|---|
| Accuracy | 0.77 | 0.78 | 0.77 | 0.81 |
| F1-score | 0.59 | 0.61 | 0.64 | 0.68 |
| Precision | 0.62 | 0.72 | 0.80 | 0.75 |
| Recall | 0.64 | 0.61 | 0.65 | 0.70 |
| FPR | 0.07 | 0.07 | 0.08 | 0.06 |
| FNR | 0.36 | 0.39 | 0.35 | 0.30 |

The ensemble between two neural networks and a gradient boosting machine is our overall best performing model. We therefore looked into deeper detail of this model. In Table 15 we see the confusion matrix of the model showing good overall performance for all the classes. Two major problems in classification is that the R2L and the DoS classes have a bias towards the normal class. Especially the R2L is often wrongly classified which is also apparent from the ROC curves shown in Figure 14.

Table 15: Confusion matrix for the ensemble model trained on CFS SMOTE

| | | Predicted Class | | | | |
|---|---|---|---|---|---|---|
| | | DoS | Normal | Probe | R2L | U2R |
| True Class | DoS | **1852** | 318 | 25 | 43 | 0 |
| | Normal | 13 | **2829** | 63 | 5 | 4 |
| | Probe | 53 | 98 | **570** | 3 | 2 |
| | R2L | 0 | 697 | 3 | **202** | 14 |
| | U2R | 0 | 6 | 0 | 0 | **14** |

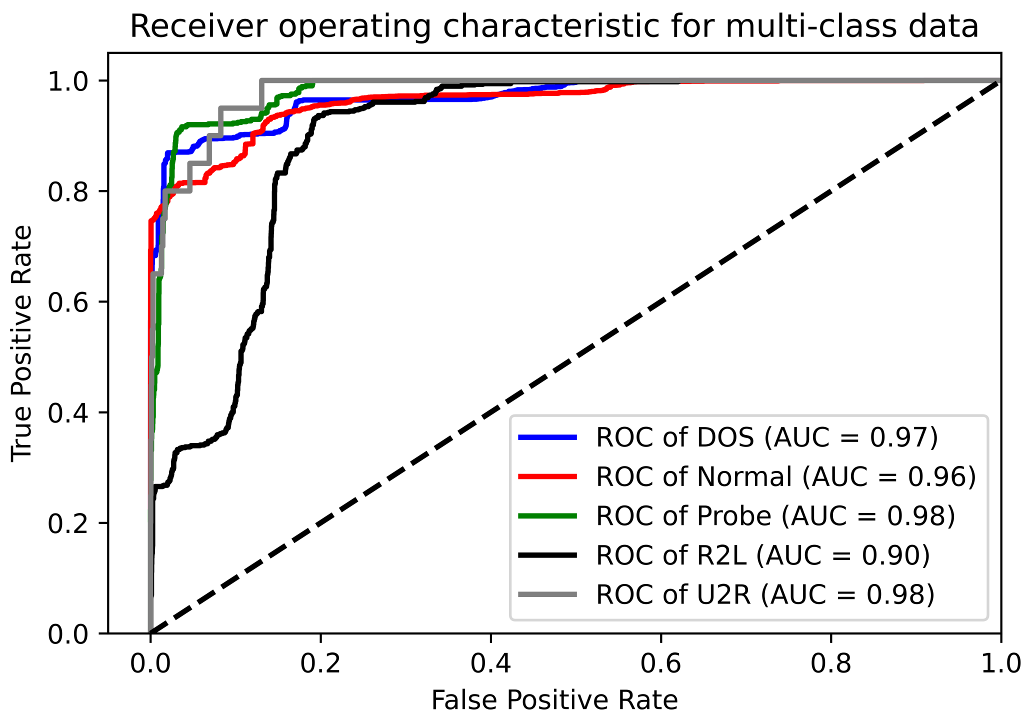

Figure 14: ROCs for ensemble trained on the CFS SMOTE data set

## 9.7 Results in Real World Application

When deploying a model in a real world application, it is crucial to be aware of its detailed behavior to every situation. For such an investigation, the arithmetic average of the metrics is no longer sufficient.

Table 16: FNR and FPR for the ensemble trained on the CFS SMOTE data set

| | Normal | DoS | Probe | R2L | U2R |
|---|---|---|---|---|---|
| FPR | 0.278 | 0.015 | 0.015 | 0.009 | 0.003 |
| FNR | 0.029 | 0.172 | 0.215 | 0.767 | 0.300 |

From Table 16 the individual false negative rates and false positive rates are computed. Notice that the model misses over 75% of all R2L classes, but for all other attack types it detects around 75% of all attacks. This means that R2L is a weak point for our model, and this must be considered when it is deployed in combination with other security measures.

To get an idea of how useful the model is in real world application we read from Table 17 that our model has a FNR for attacks on 36%. Therefore it would never work as a stand alone system. On the other hand the FPR for attacks is below 1% meaning that when the system says something is an attack it probably is.

Table 17: FNR and FPR for attack and normal

| | Normal | Attack |
|---|---|---|
| FPR | 0.278 | 0.008 |
| FNR | 0.029 | 0.363 |

These observations show that a potential usage of our system should be in combination with another measure. Such a measure could be a misuse-based detection filter, because it complements our model well. The misuse-based detection filter would have priority, if it said a package was malicious, and the anomaly said it was benign. If the anomaly-based filter said a package was malicious, and the misuse-based detection filter did not, the anomaly based would have priority.

This way the high miss rate of our system would be reduced for known attacks by the misuse-based detection filter. If our anomaly-based filter detected an unknown attack the system as a whole would still detect it with a very low false alarm rate.

# 10 Conclusion

The goal with this thesis was to provide the reader with an overview of conventional machine learning in anomaly-based intrusion detection. We have explored the following preprocessing techniques: SMOTE as a mean for handling imbalance correction of data, and CFS, FCBF and Consistency-based feature selection. Both SMOTE and CFS showed good results and by combining them we obtained the best performing data set.

We trained a neural network, a random forest, a gradient boosting machine and an ensemble of neural networks and a gradient boosting machine on the data sets obtained by the above mentioned preprocessing techniques.

The best performing model was an ensemble with 2 neural networks and a gradient boosting machine trained on the data set, which had been through both SMOTE imbalance correction and CFS feature selection. This model performed significantly better than a similar model trained on the raw data set. We obtained an accuracy of 81% and a F1-score of 68%, which is an improvement of respectively 10% points and 24% points compared to training the model on the raw data set.

Due to a high miss rate for attack classes we concluded that an anomaly-based intrusion detection system only based on our best performing model would not comprise a satisfactory intrusion detection system. With that being said, we proposed a combination of a misuse-based model and our best performing model. The misuse filter would be able to detect all known attacks with a very little error. Our model would then be able to detect unknown attacks seamlessly in the background due to a false alarm rate under 1% for attack classes.

# References

[1] W. L. Ali A. Ghorbani and M. Tavallaee, *Network Intrusion Detection and Prevention: Concepts and Techniques.* Springer. page 116.

[2] N. Araújo, R. d. Oliveira, A. A. Shinoda, and B. Bhargava, "Identifying important characteristics in the kdd99 intrusion detection dataset by feature selection using a hybrid approach," 2010.

[3] K. Woods and K. W. Bowyer, "Generating roc curves for artificial neural networks," June 1997.

[4] S. Morgan, "Global cybercrime damages predicted to reach $6 trillion annually by 2021," December 2018. Accessed: 25/04-2020 | Link: https://cybersecurityventures.com/cybercrime-damages-6-trillion-by-2021/.

[5] W. L. Ali A. Ghorbani and M. Tavallaee, *Network Intrusion Detection and Prevention: Concepts and Techniques.* Springer. page 55.

[6] H. C. T. Kwangjo Kim, Muhamad Erza Aminanto, *Network Intrusion Detection using Deep Learning: A Feature Learning Approach.* Sorunger, 2018. page 2 through 11.

[7] W. L. Ali A. Ghorbani and M. Tavallaee, *Network Intrusion Detection and Prevention: Concepts and Techniques.* Springer. page 62.

[8] M. Tavallaee, E. Bagheri, W. Lu, , and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," 2009.

[9] D. R. Wilson and T. R. Martinez, "Improved heterogeneous distance functions," 1997.

[10] H. He, Y. Bai, E. A. Garcia, , and S. Li, "Adasyn: Adaptive synthetic sampling approach for imbalanced learning," 2008.

[11] T. M. Cover and J. A. Thomas, *Elements Of Information Theory.* John Wiley  Sons, 2006.

[12] C. Sunil Kumar and R. Rama Sree, "Application of ranking based attribute selection filters to perform automated evaluation of descriptive answers through sequential minimal optimization model," October 2014.

[13] L. Yu and H. Liu, "Efficient feature selection via analysis of relevance and redundancy," April 2004.

[14] M. A. Hall, *Correlation-based Feature Selection for Machine Learning.* PhD thesis, The University of Waikato, April 1999.

[15] P. Grünwald, "A tutorial introduction to the minimum description length principle," 2004.

[16] J. M. B. A. Arauzo-Azofra and J. L. Castro, “Consistency measures for feature selection,” February 2007.

[17] M. Dash and H. Liu, “Consistency-based search in feature selection,” March 2003.

[18] J. Kozak, *Decision Tree and Ensemble Learning Based on Ant Colony Optimization*. Springer, 2019.

[19] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer, second ed., 2017.

[20] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, “Lightgbm: A highly efficient gradient boosting decision tree,” 2017.

[21] W. M. C. Katarzyna Janocha, “On loss functions for deep neural networks in classification,” 2017.

[22] A. G. S. M. Chigozie Enyinna Nwankpa, Winifred Ijomah, “Activation functions: Comparison of trends in pratice and research for deep learning,” 2018.

[23] J. L. B. Diederik P. Kingma, “Adam: A method for stochastic optimization,” 2015.

[24] G. Vanwinckelen and H. Blockeel, “On estimating model accuracy with repeated cross-validation,” 2012.

[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[26] A. Tharwat, “Classification assessment methods,” August 2018.

[27] H. Liu and B. Lang, “Machine learning and deep learning methods for intrusion detection systems: A survey,” October 2019.

[28] T. Fawcett, “An introduction to roc analysis,” December 2005.

[29] CISA, “Understanding denial-of-service attacks.” https://www.us-cert.gov/ncas/tips/ST04-015.

[30] J. K. Andreas Björklund and S. Westergren, “Dll spoofing in windows.” https://www.it.uu.se/edu/course/homepage/sakdat/ht05/assignments/pm/programme/DLL_Spoofing_in_Windows.pdf.

# A Appendix

Table 18: Specifics on features in NSL-KDD

| Feature Number | Feature Name | Feature Type | Data Type |
|---|---|---|---|
| x1 | Duration | Basic Feature | Numeric |
| x2 | Protocol Type | Basic Feature | Nominal |
| x3 | Service | Basic Feature | Nominal |
| x4 | Flag | Basic Feature | Nominal |
| x5 | Src Bytes | Basic Feature | Numeric |
| x6 | Dst Bytes | Basic Feature | Numeric |
| x7 | Land | Basic Feature | Numeric |
| x8 | Wrong Fragment | Basic Feature | Numeric |
| x9 | Urgent | Basic Feature | Numeric |
| x10 | Hot | Basic Feature | Numeric |
| x11 | No. Failed Logins | Content Feature | Numeric |
| x12 | Logged In | Content Feature | Numeric |
| x13 | No. Comprimsed | Content Feature | Numeric |
| x14 | Root Shell | Content Feature | Numeric |
| x15 | Su Attempted | Content Feature | Numeric |
| x16 | No. Root | Content Feature | Numeric |
| x17 | No. File Creations | Content Feature | Numeric |
| x18 | No. Shells | Content Feature | Numeric |
| x19 | No. Access Files | Content Feature | Numeric |
| x20 | No. Outbound Cmds | Content Feature | Numeric |
| x21 | Is Host Login | Content Feature | Numeric |
| x22 | Is Guest Login | Content Feature | Numeric |
| x23 | Count | Traffic Feature(Same Service) | Numeric |
| x24 | Srv Count | Traffic Feature(Same Service) | Numeric |
| x25 | Serror Rate | Traffic Feature(Same Service) | Numeric |
| x26 | Srv Serror Rate | Traffic Feature(Same Service) | Numeric |
| x27 | Rerror Rate | Traffic Feature(Same Service) | Numeric |
| x28 | Srv Rerror Rate | Traffic Feature(Same Service) | Numeric |
| x29 | Same Srv Rate | Traffic Feature(Same Service) | Numeric |
| x30 | Diff. Srv Rate | Traffic Feature(Same Service) | Numeric |
| x31 | Srv Diff. Host Rate | Traffic Feature(Same Service) | Numeric |
| x32 | Dst Host Count | Traffic Feature(Same Host) | Numeric |
| x33 | Dst Host Srv Count | Traffic Feature(Same Host) | Numeric |
| x34 | Dst Host Same Srv Rate | Traffic Feature(Same Host) | Numeric |
| x35 | Dst Host Diff Srv Rate | Traffic Feature(Same Host) | Numeric |
| x36 | Dst Host Same Src Port Rate | Traffic Feature(Same Host) | Numeric |
| x37 | Dst Host Srv Diff Host Rate | Traffic Feature(Same Host) | Numeric |
| x38 | Dst Host Serror Rate | Traffic Feature(Same Host) | Numeric |
| x39 | Dst Host Srv Serrer Rate | Traffic Feature(Same Host) | Numeric |
| x40 | Dst Host Error Rate | Traffic Feature(Same Host) | Numeric |
| x41 | Dst Host Srv Rerror Rate | Traffic Feature(Same Host) | Numeric |

## B Appendix

Table 19: Distribution of classes in train and test set

| **Data Set** | **Number of Records:** | | | | | |
|---|---|---|---|---|---|---|
| | **Total** | **Normal** | **DoS** | **Probe** | **U2R** | **R2L** |
| NSL-KDD Train+ | 125973 | 67343(53%) | 45927(37%) | 11656(9.11%) | 52(0.04%) | 995(0.85%) |
| NSL-KDD Test+ | 22544 | 9711(43%) | 7458(33%) | 2421(11%) | 200(0.9%) | 2654(12.1%) |

## C Appendix

Table 20: Mapping of attacks types [red attacks only appear in testing set]

| **Attack Type** | **Attack Name** |
|---|---|
| DoS | land, pod, teardrop, back, neptune<br>smurf, apache2, worm, mailbomb, processtable<br>udpstorm |
| Probe | portsweep, ipsweep, satan, nmap<br>mscan, saint |
| R2L | spy, phf, multihop, ftp-write, imap, warezmaster<br>guesspasswd, warezclient, snmpguess, sendmail<br>xlock, xsnoop, named, httptunnel, snmpgetattack |
| U2R | perl, loadmodule, rootkit, buffer-overflow<br>xterm, sqlattack, ps |

## D Appendix

**Denial of Service [DoS]:** A DoS attack is an attack that temporarily or indefinitely makes a service unavailable to its intended users. It is described in [29] An example of this could be a SYN-flood attack. A SYN-flood attack is when some one repeatedly sends SYN requests to a server to fill up the memory of the server, and hence denying other users to get access.

**Probing:** Network probes are described in [5]. Network probes are often network scans where the attacker is searching for vulnerabilities. These attacks are not damaging in themselves, but often prior to other more harmful attacks. Therefore we want to avoid the probing to take place. An example of a probing attack could be an IPSweep or a PortSweep, which could be done with the tool nmap.

**Remote to Local [R2L] and User to Root [U2R]:** R2L and U2R are both examples of a privilege escalation attack which is described in [5]. R2L is known as a horizontal privilege escalation attack while U2R is known as a vertical privilege escalation attack.

A horizontal privilege escalation attack is when a non-user becomes a user of the system when it is not intended. An example of this could be a dictionary attack. A dictionary attack is when the attacker knows a username to the system and then tries by brute force to guess the password.

A vertical privilege escalation attack is when a user of the system acquire super user or root privileges to the system without permission. An example of a vertical privilege escalation attack is DLL-hijacking which is a known type of loadmodule attack. Instead of including the same routines in multiple executables, operating systems use shared libraries to load shared code from. Windows has a special implementation of this called Dynamic-Linked-Library [DLL]. By using the specifics of the loading mechanism, hackers can place malicious code disguised as standard routines. When the loading algorithm includes the code into an executable, the malicious code can then be executed with root privileges [30].

# E Appendix

The following is the text included in the readme.txt included in the attached .zip-folder.

The following text-document describes the pipeline for the empirical work conducted in the bachelor thesis. All the important code sections and data sets are included in this folder.

We have used the programming languages R and Python to process and model the data.

The following is a list of all the folders it contains, and which purpose they serve:

**1. Data** It contains 2 folders: NSL-KDD and NSL-KDD-SMOTE. The latter is a folder contained the over-sampled data done by SMOTE, and the first contains the original NSL-KDD data sets.

**2. Preprocessing** It contains 3 folders and a jupyter notebook. The jupyter notebook is used to modify the raw data set and the imbalance corrected data set according to the selected features. The features were found with FeatureSelection.R, which is located inside "Feature Selection". The folder Imbalance Correction contains the code for running SMOTE.

**3. Hyperparameter Fitting** It contains 3 folders: Configurations, Script for Hyperparameter Fitting, and ML-tools. Configurations contain the results from trying the different configurations on neural networks. The folder "Scripts for Hyperparameter Fitting" contains the scripts used to do hyperparameter fitting on all the models. Since RF, GBM, and the ensembles were mainly fitted with the sklearn, they do not include all the raw outputs. However, the jupyter notebooks should contain the best configurations, which are the ones stated in the thesis. The last ML-tools contain custom functions to calculate metrics and log each trial on the neural network-session.

**4. Trained Models** Contains the models for NNs, RF, GBM, and the best ensemble in folder depending on which data set it was trained on.

**5. Result Extraction** Contains a Jupiter Notebook which loads all the data and the respective models, and then it predicts on the given test data from NSL-KDD99. You should be able to read all the results without running anything. These outputs lay the foundation for our result section.