UNIVERSITY OF HASSELT

BACHELOR THESIS

# Machine learning techniques for flow-based network intrusion detection systems

*Advisor:*
Prof. Dr. Peter QUAX
Prof. Dr. Wim LAMOTTE
*Mentor:*
Bram BONNE
Pieter ROBYNS

*Author:*
Axel FAES

*Bachelorproef voorgedragen tot het behalen van de graad van bachelor in de informatica/ICT/kennistechnologie*

*A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science*

*in the*

Networks and Security
Computer Science

Academiejaar 2015-2016
June, 2016

universiteit
hasselt

KNOWLEDGE IN ACTION

# *Acknowledgements*

I would like to thank my mentor, Bram Bonne, with whom I had weekly meetings. He has given feedback on this thesis and had a lot of suggestions regarding the machine learning chapters. I would like to thank Cegeka, to provide me with a dataset on which I could test the implementation. Thanks also go towards my promotor professor Peter Quax and co-promotor professor Wim Lamotte for their advise and providing me with the EDM dataset.

From my classmates, I would like to thank Matthijs Kaminski for his support throughout the bachelor. I would also like to thank Luuk Raaijmakers for reading over my thesis.

# *Abstract*

Large data centers are storing and sending more and more data. In order to check whether the network traffic does not contain intrusions, an intrusion detection system is used. Such a system analyses data from the network and gives an alert if it finds an intrusion. Since data centers have so much data traffic, it is difficult to process everything. That's where IP flows come in the picture. They are aggregated from packet data but do not contain any information about the payload data. This thesis explains which attacks can be detected and how IP flows can be used for intrusion detection. It would also be cost-efficient if an intrusion detection system could operate automatically and detect attacks with a high probability. For this machine learning can be used. Machine learning is a type of Artificial Intelligence which allows programs to learn and find patterns within data. However, there are many different types of machine learning. This thesis gives an introduction to machine learning concepts and gives an overview of different machine learning algorithms such as Support Vector Machines and K-Nearest Neighbors. An explanation is given on how these algorithms can be used in an intrusion detection system. The algorithms are evaluated on different datasets which consist of both labeled training data and unlabeled real world data. The evaluation is done using learning curves and F-scores. In the evaluation it is found that supervised learning gives better and more detailed predictions as compared to unsupervised learning. K-Nearest Neighbors gives the best results among the tested supervised learning algorithms. The results show that machine learning is a viable option to detect intrusions using IP flows. It can also be noted that using extra information such as the TCP flags are a usefull addition, and increase the performance significantly.

# Contents

# Nederlandstalige samenvatting (Dutch Summary)

Er zijn vele soorten gevaren op het internet, waaronder malware en DDOS aanvallen. Een netwerk kan beveiligd worden tegen zulke aanvallen met behulp van een intrusie detectie systeem. Een intrusie detectie systeem kan intrusies detecteren en genereert een alert wanneer het een intrusie detecteert.

Deze intrusie detectie systemen bekijken een netwerken en analyseren alle trafiek. Voor grote datacenters wordt dit echter een zware taak. Er gaat enorm veel data doorheen het netwerk van een datacenter. Standaard intrusiesystemen kunnen het niet aan om dan alle trafiek volledig na te kijken. Een manier om dit op te lossen is door IP flows te gebruiken. Dit is geagregeerd van packet data. Het gebruiken van IP flows zorgt ervoor dat een intrusie detectie systeem alle trafiek kan nakijken. Hierbij treedt dan de vraag op ofdat via IP flows alle intrusies gedetecteerd kunnen worden.

Intrusie detectie systemen vereisen ook veel manueel onderhoud. Hieraan hangt natuurlijk ook een hoge kost. Deze thesis tracht dan ook te bekijken ofdat een intrusie detectie systeem out-of-the-box een goede performance kan hebben. Dit wordt gedaan via machine learning algoritmes. Dit zijn algoritmes die kunnen leren van data en patronen kunnen herkennen. Dit lijkt goed toepasselijk voor het probleem van intrusie detectie, dit zal deze thesis ook bekijken, alsook welke algoritmes wel of niet werken. Een andere belangrijke vraag is natuurlijk ofdat zo een systeem ook toepasbaar is in een real-world scenario.

## 0.1 Intrusion detection systems

Intrusie detectie systemen of IDS's kunnen intrusies detecteren op verschillende manieren. Een intrusie kan ook een aanval of een anomalie genoemd worden. Een intrusie detectie systeem kan ofwel het netwerk monitoren, ofwel systeem activiteiten monitoren. Een intrusie detectie systeem dat netwerkgedrag monitort wordt een netwerk-gebasseerd intrusie detectie systeem genoemd. Een intrusie detectiesysteem dat systeem activieten monitort wordt een host-gebasseerd intrusie detectie systeem genoemd. Deze thesis implementeerdt een netwerk-gebasseerd intrusie detectie systeem.

Detectie kan gebeuren met behulp van twee methodes. De eerste methode is een signature-gebasseerde methode. Hierbij wordt een database bijgehouden waarin signatures zitten die aantonen hoe intrusies eruit zien. Het intrusie detectie systeem bekijkt dan telkens ofdat de signature van een packet of flow matched met een signature van de database. De andere methode is een anomalie-gebasseerde methode. Hierbij stelt het intrusie detectie systeem een statistisch model op wat normaal netwerkgedrag voorstelt. Alles wordt vergeleken met dit model en als er een afwijking opgemerkt wordt, wordt dit gezien als een intrusie.

## 0.2 Attack classification

Een intrusie detectie systeem kan intrusies detecteren. Om het intrusie detectie systeem efficienter te maken, moet exacte classificatie van een intrusie mogelijk zijn. Er wordt een onderscheid gemaakt tussen twee soorten intrusies. Er zijn interne en externe intrusies. Onder de externe intrusies behoren DDOS, Brute-force aanvallen, vulnerability scans, man in the middle attacks en buffer overflows. Interne intrusies kan ook malware genoemd worden. Er zijn verschillende soorten malware. Er zijn virusen, trojan horses, worms en botnets.

Via IP flows kunnen niet alle soorten intrusies gedetecteerd worden. Meer bepaald, een intrusie detectie systeem dat IP flows gebruikt kan enkel DDOS, vulnerability scans, worms en botnets detecteren. Andere soorten aanvallen gebruiken niet zozeer netwerk communicatie. Echter kunnen er onder bepaalde omstandigheden toch andere intrusies gedetecteerd worden. Bijvoorbeeld, als er geweten is dat virussen

altijd vanuit dezelfde bron verzonden worden, kan er toch geweten worden ofdat een flow hoogst-waarschijnlijk een virus bevat.

## 0.3   IP Flows

IP flows zijn geaggregeerd van alle packet data die door een netwerk gaan. Een flow is niet hetzelfde als een TCP connectie. Een flow wordt gezien als alle communicatie tussen twee apparaten. Flows worden geidentificeerd met een (source_IP, destination_IP, protocol) tuple.

Niet alle intrusies kunnen gedetecteerd worden omdat flows geen informatie bevatten over de pay-load die in de flow voorkomt. Enkel het aantal paketten en het aantal bytes in de flow is gekend. Ook de source en destination poorten zijn gekend. De start tijd en de duratie van de flow worden ook bijge-houden.

Een flow exporter is een programma dat flows opbouwd vanuit het netwerkverkeer. Sommige flow exporters staan ook toe extra informatie bij te houden in de flow. Bijvoorbeeld het aantal TCP SYNs in een flow kan bijgehouden worden.

## 0.4   Machine learning

Machine learning vormt een subdomein binnen de Informatica. Het is een soort van Artificial Intelli-gence dat programma's toelaat om zelf te leren en patronen te herkennen in data. Leren betekent dat een algoritme data getoond moet worden. Dit algoritme gebruikt deze data om een model op te stellen waarmee het voorspellingen kan doen voor nieuwe data.

Er zijn twee soorten machine learning algoritmes. Er zijn supervised learning algoritmes en unsuper-vised learning algoritmes. Supervised learning gebruikt trainingsdata die gelabeled is. In unsupervised learning moet het algoritme zelf structuur zoeken binnen de trainingsdata.

Een machine learning algoritme probeert een proces te modelleren. Bijvoorbeeld, een machine learning algoritme kan gebruikt worden om te voorspellen wat het studiegeld volgend jaar gaat zijn. Een heel simpele manier om dit te doen is door het algoritme te leren wat de kost de afgelopen jaren was. Het algoritme stelt vervolgens een model op en tracht de kost voor volgend jaar te voorspellen gebasseerd op de data van de vorige jaren.

Het model dat het algoritme opstelt word een hypothese genoemd. Om een proces te modeleren ge-bruikt een machine learning algoritme features van het te modeleren proces. Een feature kan alles zijn. In voorafgaand voorbeeld zijn de features, een jaartal en een kost: $(jaar, kost)$.

De hypothese wordt wiskundig voorgesteld als een functie zoals: $H_0(x) = \theta_0 * x_0 + \theta_1 * x_1 + ... + \theta_n * x_n$. Hierbij zijn $(x_0, x_1, ..., x_n)$ de features. $(\theta_0, \theta_1, ..., \theta_n)$ zijn onbekendes. Hiervoor moet een goede waarde gevonden worden opdat de functie $H_0(x)$ een goede voorspelling kan doen. Dit gebeurt door een kost functie. Een kost functie berekend hoeveel de waarde van $H_0(x)$ en de eigenlijke waarde verschillen. Deze kost functie moet geminimaliseerd worden. In de thesis worden deze concepten dieper uitgelegd, alsook welke problemen er kunnen optreden en hoe ze opgelost kunnen worden

Er kunnen verschillende soorten algoritmes gebruikt worden. De twee meest veelbelovende algoritmes zijn Support Vector Machines en K-Nearest Neighbors. Support vector machines zijn een supervised learning algoritme. Het algoritme tracht om classificatie uit te voeren. Het tracht een goede opsplitsing te vinden tussen de verschillende klasses die vertegenwoordigd zijn in de trainingsdata.

K-Nearest Neighbors is ook een supervised learning algoritme. De werking is echter wel ander dan

Support Vector Machines. Support Vector Machines werken via de meer traditionele manier via een hypothese en kost functie. K-Nearest Neighbors of KNN doet dit niet. Wanneer KNN een voorspelling moet uitvoeren op een data punt, wordt er gekeken welke $k$ datapunten vanuit de trainingsdata het dichtste liggen bij het te voorspellen datapunt. Er wordt dan gekeken welke klasse het meeste voorkomt in deze $k$ punten en dit is de voorspelling.

Een belangrijk onderdeel is kijken hoe goed een algoritme werkt. Dit kan gedaan worden via precision P, recall R en de F-score. Deze concepter zijn simpel uit te leggen binnen binaire classificatie. Een data-punt is geklassificeerd als positive of negative. Een hoge precision stelt dat samples die geklassifiseerd zijn als positive ook eigenlijk positive zijn. Een hoge recall stelt dat samples die positive zijn ook geklas-sificeerd zijn als positive. De F-score is een berekening met de precision en recall: $(2 * P * R)/(P + R)$. Hoe hoger deze score is, hoe beter het algoritme werkt.

## 0.5 Implementatie

In de implementatie is gebruik gemaakt van *scikit-learn*. Dit is een robuuste machine learning library voor Python. Het is gebouwd bovenop *NumPy*, *SciPy* en *Matplotlib*. Deze library implementeerdt de meeste algoritmes die besproken worden in deze thesis. Enkel neurale netwerken zijn niet geïmple-menteerd in deze library. Deze zijn wel geïmplementeerd in de unstable versie van deze library.

De implementatie is opgebouwd uit verschillende modules. Hierdoor kunnen nieuwe componenten simpel toegevoegd worden. De eerste module is de machine learning module. Deze bevat alle machine learning algoritmes die gebruikt zijn. Er is ook een feature module. Deze module bevat klasses die gebruikt worden om features te verkrijgen uit de IP flows.

Een loader module bevat alle klasses die nodig zijn om data te laden uit de verschillende data sets. De training module gebruikt de loaders en selecteert welke data gebruikt wordt om een algoritme mee te trainen. Een results module verkrijgt alle input en kan deze loggen of visualiseren.

Om te bepalen welke modules gebruikt worden in de implementatie word een JSON config file ge-bruikt. Deze config file definieert welke modules gebruikt moeten worden alsook welke datasets ge-bruikt moeten worden.

Er zijn vier verschillende datasets gebruikt. Elke dataset dient een ander doel. Twee datasets zijn ge-bruikt voor een machine learning algoritme te trainen op respectievelijk interne en externe intrusies. Deze datasets zijn ook gebruikt om te verifiëren dat het machine learning algoritme werkt via de F-score. De andere datasets zijn gebruikt om de machine learning algoritmes te testen op real-world data.

De eerste dataset is de CTU-13 dataset. Dit is een gelabelde dataset die interne intrusies bevat. De klassificatie in deze dataset is zeer accuraat. Er zijn bijvoorbeeld labels voor botnets, google analytics en windows updates. De volgende dataset is de Tracelabel dataset. Deze dataset is ook gelabeld maar bevat externe intrusies. Ze is gemaakt op de Universiteit van Twente door een honeypot te plaatsen. Hierdoor bevat de dataset slechts een klein aantal normale flows die geen intrusies zijn.

Er is een dataset verkregen van het EDM. Deze bevat ongelabelde data die gaat vanaf 18 Februari tot 24 maart 2016. Er is data van elke dag vanaf 10u tot 24u. Er is ook een dataset verkregen van Cegeka. Deze dataset gaat over drie dagen, van 3 April tot 5 April. Deze bevat ook firewall logs zodanig dat er gekeken kan worden ofdat de klassificatie correct gebeurt.

Bijna alle attributen van een flow zijn gebruikt als features. Enkel de start tijd is niet gebruikt. Dit opdat de trainingdata niet genoeg data bevat over een een voldoende grote tijdsspanne. Alle features

zijn als continue data voorgesteld. De meeste attributen van een flow zoals de IP adressen, aantal paketten en bytes zijn al continu, maar de source en destination poort, alsook het protocol is discrete data.

De poorten zijn omgezet naar een feature die bekijkt ofdat de gebruikte poort een van de 1024 veelgebruikte poorten is of niet. Voor elk protocol is een andere binaire feature gebruikt.

## 0.6   Evaluatie

Om de machine learning algoritmes te evalueren, is er gebruik gemaakt van learning curves en de F-score. Er is tevens een baseline opgestelt om de machine learning algoritmes mee te vergelijken. Deze baseline is een algoritme dat willekeurig voorspellingen maakt. De learning curve is gebruikt om te bepalen ofdat bij het algoritme overfitting of underfitting optreedt. De F-score is gebruikt om de eigenlijke performance te meten.

Tijdens de evaluatie zijn zowel supervised learning algoritmes als unsupervised learning algoritmes gebruikt. Bij de unsupervised learning algoritmes trad er het probleem op dat er moeilijk afgeleid kon worden ofdat de flow een intrusie was of niet. One-class Support Vector Machines, een unsupervised learning algoritme, kon niet met goede accuraatheid stellen of een flow een intrusie was of niet. Het was zelfs zo dat het amper beter werkte dan een compleet willekeurig algoritme.

Supervised learning gaf veel betere resultaten. Er kon een goed evenwicht gevonden worden tussen precision en recall zodanig dat het syteem goed reageerde en niet teveel false positives gaf. Van de geteste supervised learning algoritmes is gevonden dat K-Nearest Neighbors de beste performance had.

Er kan geconcludeerd worden dat IP flows gebruikt kunnen worden om intrusies te detecteren. Een flow-gebasseerd intrusie detectie systeem zou gebruikt kunnen worden als een van de eerste lagen van verdediging van een netwerk. Zodanig kan er al een grote filtering gebeuren op de inkomende data en kunnen al veel intrusies gedetecteerd worden.

De machine learning algoritmes bleken goed te werken voor intrusie detectie systemen. Mits er goede trainingsdata beschikbaar is, kan een intrusie detectie systeem een goede performance behalen zonder manueel onderhoud. Ook zijn er goede resultaten behaald met de data van het EDM en van Cegeka. Dit toont aan dat het systeem ook in real-life scenario's gebruikt kan worden.

# Chapter 1

# Introduction

The internet is constantly growing and new network sevices arise constantly. Sensitive data is also increasingly being stored digitally. All these new services could contain security flaws which could leak private data, such as passwords or other sensitive data. This means that security flaws become more and more important since they can cause so much damage. It is not just the leaking of sensitive data that is an issue, but also protecting a computer or network against malware is important.

For example, earlier this year, the internal network of a hospital was attacked by a ransomware attack. The attackers did not get access to personal information and the hospital was able to go back to doing paperwork. But it does show that cyber attacks are a serious threat. [1]

Considering this, it becomes more important to be able to detect and prevent attacks on network systems. Intrusion detection systems are used for this purpose. An intrusion detection system can alert administrators of malicious behaviour. Intrusion detection systems can detect attacks using several different methods as explained in Section 2.

These methods are not always reliable. They may not catch every attack in which case the intrusion detection system is not that useful. They could also give alerts for events that are not malicious at all. This could desensitize administrators for the alerts an intrusion detection system gives.

In order to have good performance, most intrusion detection systems need a lot of manual maintenance. This is needed to constantly check the alerts the intrusion detection system gives and tweak the system if required. Other systems which are already well configured have a big price tag. This thesis tries to find out *whether an intrusion detection system can work out-of-the-box with an acceptable performance.*

This is done by using machine learning algorithms. These are algorithms which can learn and find patterns in input. They can find out what different types of behaviour look like. This means that they could for example learn what botnet behaviour looks like. Machine learning algorithms are further explained in Section 4. *Machine learning algorithms seems promising for the problem of automatic intrusion detection, but is this true?* However, there are different kinds of machine learning algorithms. *These different kinds of algorithms will be discussed and evaluated to see whether they are useful in anomaly detection.*

The fact that the internet is growing continuously, this means that more and more data passes through the networks of big companies and data centers. This makes it quite difficult to be able to check every piece of data that passes through the network. This could be solved by using IP Flows. These aggregate a lot of packet data into a small flow, they are further explained in Section 9. These flows are small, but also contain less information than the complete packet data. *But is it possible to still detect attacks when only the IP flows are used?*

It is not just important to find out whether attacks can still be detected when only IP flows are used. It is also important to know which kind of anomalies can be detected. *Could every type of attack be detected or is it a subset?*

These questions lead to the development of a system which uses machine learning techniques in a purely flow based intrusion detection system. Such a system would be one of the first lines of defense of a company against cyber attacks. Real world data is going to be used to test *whether the system can operate in a real life situation.*

# Chapter 2

# Intrusion detection systems

An intrusion detection system is a system which tries to determine whether a system is under attack, to detect intrusions within a system. Intrusion detection systems are often called IDS's. Intrusions can also be called attacks or anomalies. It does this by monitoring network or system activities. One way of categorizing IDS's is based on the method of detection intrusion. The first type would be monitoring the network, these are called network-based intrusion detection systems, or NIDS. When the intrusion detection system only monitors system activities, it is called a host-based intrusion detection system, or HIDS. [2]

Figure 2.1 shows the possible placements of an IDS. It can be placed before any firewall, being the first defense of a network. This is a NIDS. An IDS can also be placed within a network. This IDS can still monitor the network but it can also monitor system activities of the workstations. An HIDS is most commonly, but not always located on the device that it monitors.
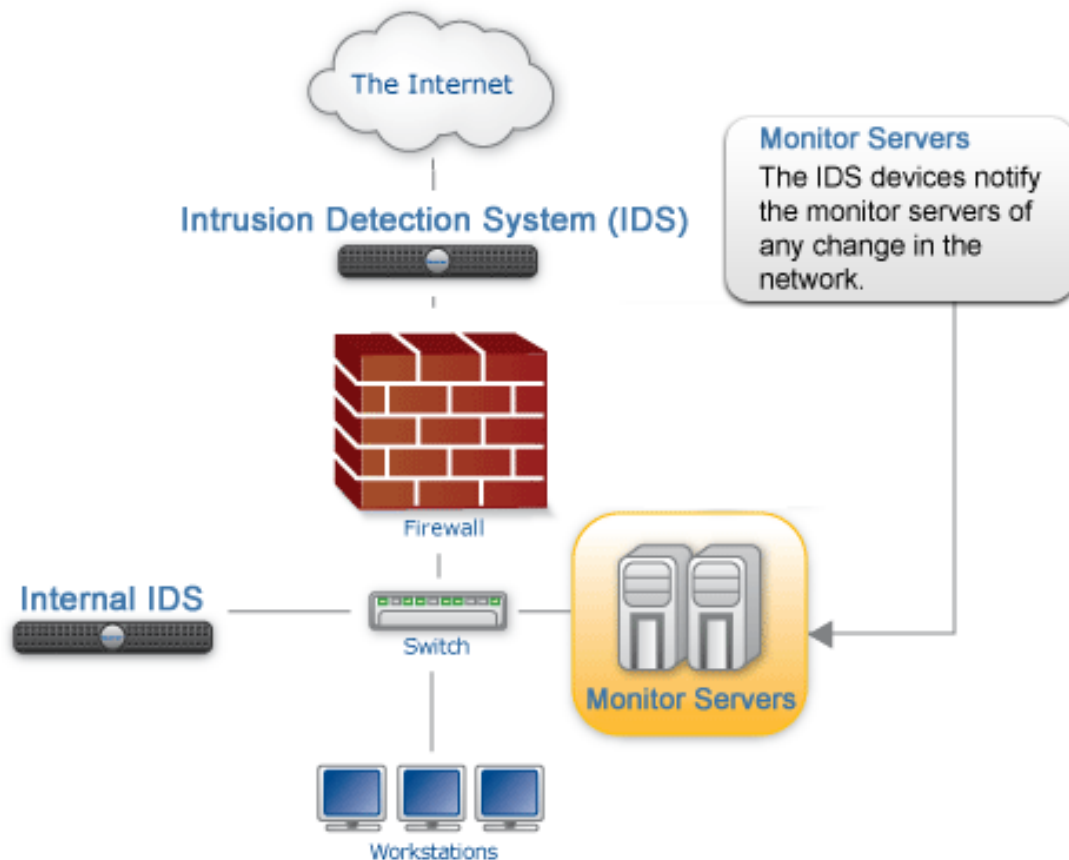
**Figure 2.1:** An IDS can for example be placed within the network or just before the network. [3]

## 2.1   Host-based Intrusion Detection Systems

Host-based intrusion detection systems are systems that monitor the device on which they are installed, or directly connected to. The way they monitor the system can range from monitoring the state of the main system through audit logs, to monitoring program execution. In this way they can be quite indistinguishable from Anti-Virus programs. [2]

Since HIDS rely so much on audit logs, they can become limited by them. If the software that is being monitored does not provide enough information in audit logs, the IDS cannot always determine intrusions succesfully. Another issue can be the sheer volume of the audit logs. Every monitored log needs to be parsed, this means that the HIDS can have a big impact on the performance of the host system if it is installed there. [4]

Another disadvantage is that any vulnerability that causes the audit files to be changed, also impacts the integrity of the HIDS. If an audit file is changed, the HIDS cannot see and detect what truly happened. [5]

## 2.2 Network-based Intrusion Detection Systems

Network-based intrusion detection systems are placed at certain points within a network in order to monitor traffic from and to devices within the network. They operate on the same concept as wiretapping. They "tap" into a network and listen to all communication that happens. [2]

Using the network data instead of audit trails such as HIDS is desirable in multiple ways. One advantage is that they do not impact the performance of programs that are using the network. It is also more difficult for an intruder to attack the IDS itself. Network traffic is always visible and cannot be changed like an audit trail. The intruder could try to minimize his network activity, but the risk is lower. NIDS are also more portable than HIDS. They monitor traffic over a network and are independent of the operating system they run on. [4]

The system can analyse the traffic using multiple techniques to determine whether the data is malicious. There are two different ways to analyse the network data. The analysis can be packet-based or flow-based.

Packet-based analysis uses the entire packet including the headers and payload. An intrusion detection system that uses packet-based analysis is called a packet-based network intrusion detection system. The advantage of this type of analysis is that there is a lot of data to work with. Every single byte of the packet could be used to determine whether the packet is malicious or not. The disadvantage is immediately obvious once we look at networks through which a lot of data passes, such as data centers. Analysing every byte is very work-intensive and near impossible to do in such environments. [6]

Flow-based analysis doesn't use individual packets but uses general aggregated data about network flows. An intrusion detection system that uses flow-based analysis is called a flow-based network intrusion detection system. A flow is defined as a single connection between the host and another device. A flow can be defined using a (source_IP, destination_IP, source_port, destination_port) tuple. However flows also contains other information. IP Flows are discussed in depth in Section 9. Because of the other information, flows can still contain a lot of information, even when compared to packets. Since flow data is much more compact than all the individual packets, it is much more feasable for data centers to use flow-based intrusion detection systems. [6] [7]

## 2.3 Intrusion Prevention Systems

An intrusion prevention system or IPS/IDPS is an intrusion detection system that also has to ability to prevent attacks. An IDS does not necessarily need to be able to detect attacks at the exact moment they occur, although it is preferred. An IPS needs to be able to detect attacks real-time since it also needs to be able to prevent these attacks. For network attacks these prevention actions could be closing the connection, blocking an IP or limiting the data throughput. [8]

The change to requiring attacks to be detected at real time can severely impact the methods that are used to detect these attacks. For example, an IDS might give an alert even though the IDS is not certain that whatever it is alerting is actually an anomaly. An IPS needs to be certain before it can take action. Otherwise the IPS might take actions which the business employing the IPS does not want. [8]

In case of a NIPS, an network-based intrusion prevention system, some advantages of being network-based are not true anymore. An NIPS needs to see all data, and preferably block network data before it reaches it's destination as seen in Figure 2.2. This means that the performance of programs using the network might be affected by the NIPS. [9]
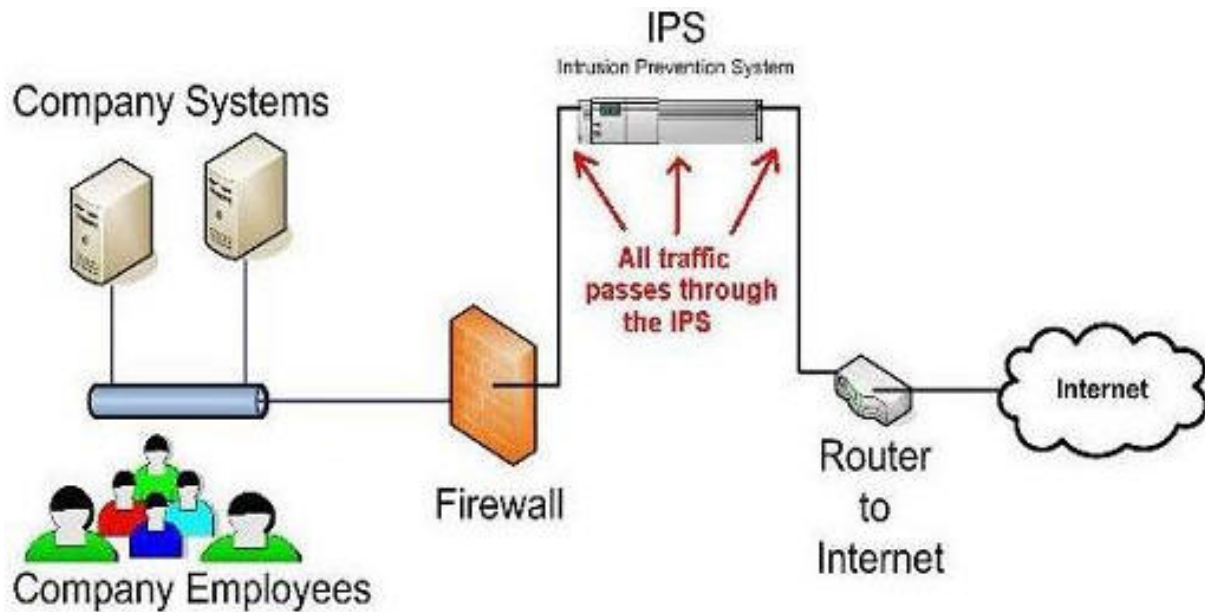
**Figure 2.2:** An Intrusion prevention system. [10]

## 2.4   Detection

There are mutliple different methods to detect intrusions. There are **Signature based methods** and there are **Anomaly Based** methods. Both of these methods have their own strengths and weaknesses.

### 2.4.1   Signature based methods
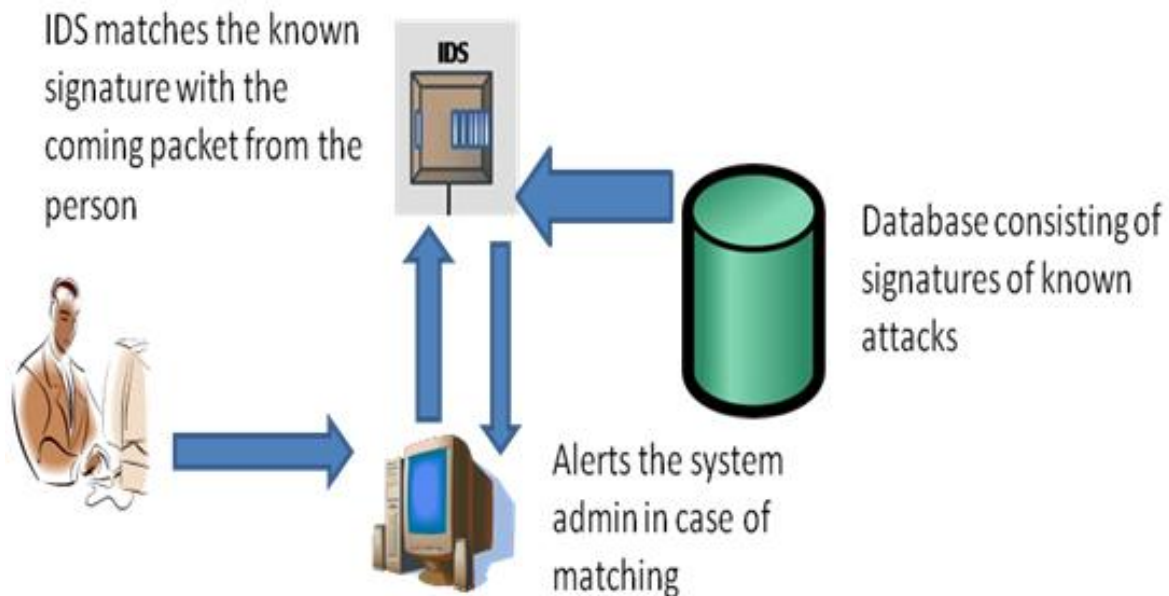


**Figure 2.3:** An Signature-based intrusion detection system. [11]

Signature based methods compare so called "signatures" with an existing database of signatures. An packet or flow record is decomposed into features that together construct a signature. If the signature of an incoming flow or packet matches with a signature in the database, it is flagged as malicious. Pseudocode of this can be seen below. Signature-based methods have little overhead in both computation and preprocessing as it only tries to match incoming signatures to known signatures in the database. Because it only compares signatures, it is easy to deploy within a network. The system does not need to learn what the traffic within a network looks like. [12]

```python
signatures = get_signatures_from_database()
while True:
    packet = get_next_packet()
    packet_signature = get_important_features(packet)

    if (signatures.contains(packet_signature):
        generate_alert(packet)
    else:
        # packet signature was not
        # a match with known signatures
        continue
```

Signature based methods are very effective against known attacks. New attacks cannot be detected unless the database is updated with new signatures. It is also possible for attackers to avoid being caught by signature based methods, only a slight modification of the "signature" is required in order to bypass the exact matching. [13]

This could be done by trying to make the network behaviour look more like normal behaviour. For

example, a botnet uses IRC communication with his master. The communication might follow certain patterns which are different from usual IRC traffic from a chat. The creator might change the botnet so that communication between the botnet and the master look similar to the usual IRC traffic. This change causes the network behaviour to be different from the signature database and not generate an alert. Updating the signature database requires a lot of technical effort, since new attacks are discovered all the time.[12] [14]

### 2.4.2   Anomaly based methods
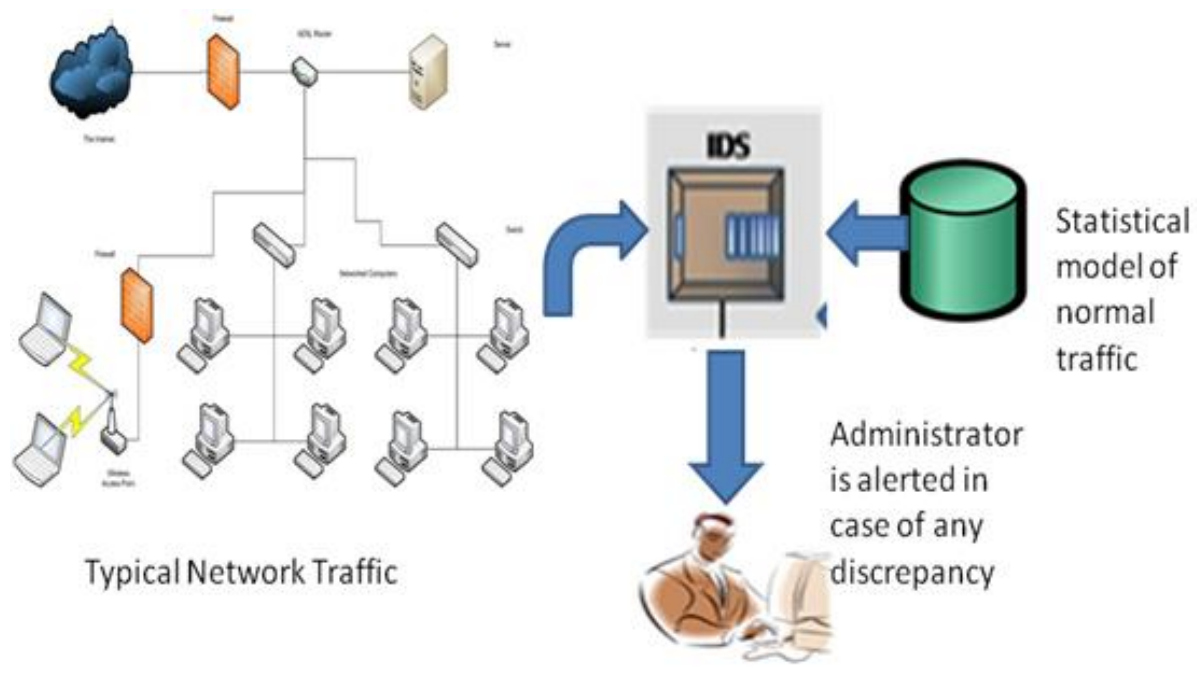


**Figure 2.4:** An Anomaly-based intrusion detection system. [11]

Anomaly based methods, also called Behaviour based methods are methods in which the IDS tries to model the behaviour of network traffic. When an incoming packet deviates from this model, it is flagged as malicious and an alert is sent. Because they use a statistical model of normal behaviour, they should be able to detect all deviations from this normal behaviour. As a result, new attacks that deviate too much from normal behaviour are detected as well. [11]

Since a model of the network traffic needs to be created, the system cannot be deployed into a network and be expected to work. The system needs to learn the behaviour of the network traffic. Problems, such as generating a lot of false positive alarms, can arise when training data includes mistakes, such as misclassifications.

Machine learning algorithms can be used as an anomaly based method. Machine learning techniques have the ability to learn from data and decide whether new data is malicous. [11]

## 2.5   Existing Intrusion detection systems

There are a lot of different intrusion detection and intrusion prevention systems on the market. Some of these systems are very expensive, others are completely open source and free.

### 2.5.1 Alienvault

*Alienvault* is a business that develops software to manage cyber attacks. They create SIEM solutions, Security Information and Event Management solutions. These are tools that provide methods to analyse security threats. IDS's are incorperated into SIEM's. [15]

The product they make is the *AlienVault Unified Security Management* (USM). USM is an all-in-one tool. USM contains a lot of tools that are required to analyse a system for cyber security threats. It contains tools that can scan and test the network for vulnerabilities.

USM supports both host-based intrusion detection and network-based intrusion detection. It also has the option to do both full packet and netflow analysis, supporting both types of NIDS systems. [16]

*Alienvault* works on both signature-based and anomaly-based intrusion detection. However, it is interesting to note that they are working on new strategies that can detect intrusions. Research is being done using neural networks to make intrusion detection more accurate. [17]

### 2.5.2 SNORT

Snort is an open-source network intrusion detection and prevention system. It is made to be very lightweight in use. It runs on UNIX derivatives and Windows. SNORT is the most widely used IDS worldwide and has become the in reality standard for the industry. SNORT can also operate on both packet level and IP flow level. However, SNORT uses a combination of signature and anomaly detection methods. [18]

SNORT uses a system based on rules. Using rules, an administrator can configure SNORT. These rules allow for both signature and anomaly detection methods. This means that SNORT itself does not work with machine learning algorithms. However, the rules can be made using machine learning algorithms. These algorithms make rules based on which traffic is classified as normal and abnormal. [19]

SNORT can, since a couple years, also be deployed as a intrusion prevention system. This is done by adding a new type of rule, a "drop" rule. A "drop" rule has higher precedence than an "alert" rule. This means that any packet that matches a "drop" rule and an "alert" rule will be dropped.

Since SNORT is open source, the internal structure can be observed. There are several components that work together to detect abnormal behaviour and to generate output in a format that is appropriate for an intrusion detection system. [18]

- Packet sniffer and decoder
- Preprocessors
- Intrusion detection engine
- Logger and alert system
- Output modules

In Figure 2.5, the structure of the SNORT components can be seen. The **packet sniffer and decoder** sniffs packets from different network interfaces. The packets are then prepared to go to the preprocessor. The **preprocessor** can already extract the most important data from packets and might even modify the data. In the **detection engine** rules are used to detect any intrusions. The rules are matched against every packet. If a packet is matched, it may be dropped or an alert can be generated, depending on the type of the rule.

Alerts can be logged to different kinds of files. This happens in the **logger and alert system**. These

could be text files or tcpdumps. The **output modules** can do different operations on the output to generate new log files. [18]
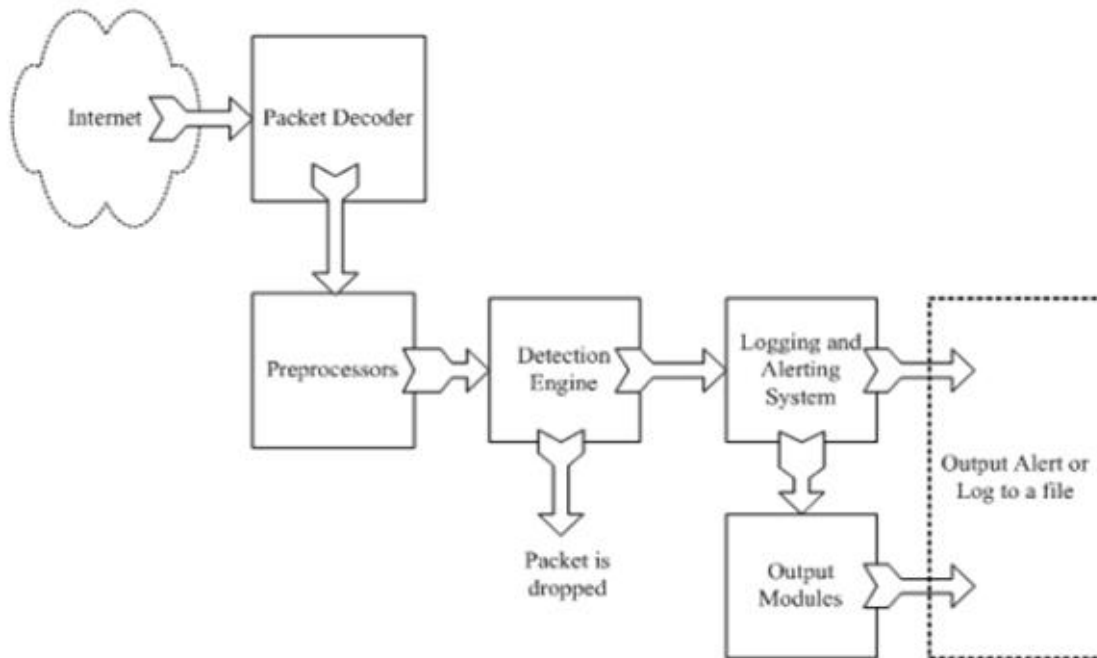


**Figure 2.5:** The structure of the SNORT IDS. [11]

### 2.5.3   MIT AI$^2$

MIT is also working on methods to use machine learning to defend against cyber attacks. In their paper "*AI$^2$: Training a big data machine to defend*", they present a new method. Their system has four components. A big data processing system, an outlier detection engine, a mechanism to obtain feedback from security analysts, and a supervised learning module. [20]

Their system tries to combine the expertise of security experts, and the speed and ability to detect new attacks of machine learning. More specifically, they use unsupervised machine learning. They prefered to use unsupervised machine learning since labeled data is rare and attacks constantly evolve. In the system they generate their own labels and use a supervised learning algorithm with these labels.

The **big data processing system** is a system that can extract features of different entities from raw data. The **outlier detection engine** is a system that uses unsupervised learning. It uses the features that have been found in the big data processing system. They use three different methods, density, matrix decomposition, or replicator neural networks. [20]

The output of this unsupervised system is processed and shown to a **security analyst**. The security analyst can verify or refute the output. The feedback is fed to a **supervised learning algorithm**. The supervised learning algorithm learn a model that can use this feedback to better predict whether any new event is normal or abnormal. With more feedback, the system becomes more and more correct. The flow of the system can be seen in Figure 2.6. [20]
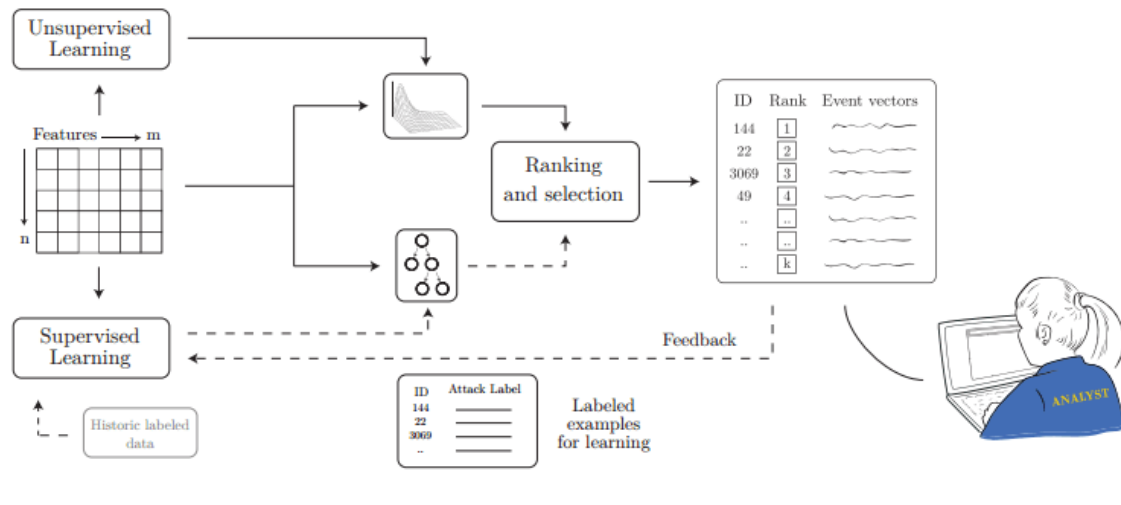
**Figure 2.6:** The structure of AI2 system. [21]

Their system has been tested by monitoring a large web-scale platform. This platform generated millions of log lines per day. The monitoring lasted three month and generated 3.6 billion log lines. They could detect 85 percent of attacks and reduced the amount of false positives by 5 times since the previous implementation. [21]

# Chapter 3

# Attack Classification

An intrusion detection system can use multiple methods to detect malicious behaviour. In order to make the IDS as effective as possible, the exact classifications of malicious behaviour that can be detected need to be known. Classification can happen on several different ways. For example they can be classified using similar behaviour or they could be classified using similar damage caused. [22]

## 3.1 Classification

A usefull classification is to first make a distinction between internal and external malicious behaviour. This makes it easier for humans to understand. The IDS itself can work with different kind of classifications. However, the IDS has to communicate with an administrator about the detections. A distinction between internal and external malicious behaviour is easier to understand.

The exact classifications are not mutually exclusive. Some types of malicious behaviour can be both internal and external. The exact classification needs to be much deeper than just internal and external. Every type of malicious behaviour is identified by different characteristics. Knowing these characteristics is useful to be able to tweak the IDS to make identification more effective. [23]

## 3.2 External abnormal behaviour

External abnormal behaviour consists of different kind of attacks on a systems. There are many different type of attacks. There are **Physical attacks**, **Buffer overflows**, **Distributed Denial of Service**, **Brute-force attacks**, **Vulnerability scans** and **Man in the middle attacks**.

### 3.2.1 DDoS

**DDOS** or Distributed Denial of Service attacks are attacks which attempt to make a network resource temporarily or permanently unavailable for the users of that resource. An attack could happen by flooding a system with TCP SYN packets. In Figure 3.1 an example can be seen. The attacker uses some Master Nodes to control a number of Daemon Nodes. These Daemon Nodes are the computers that actually carry out the attack.
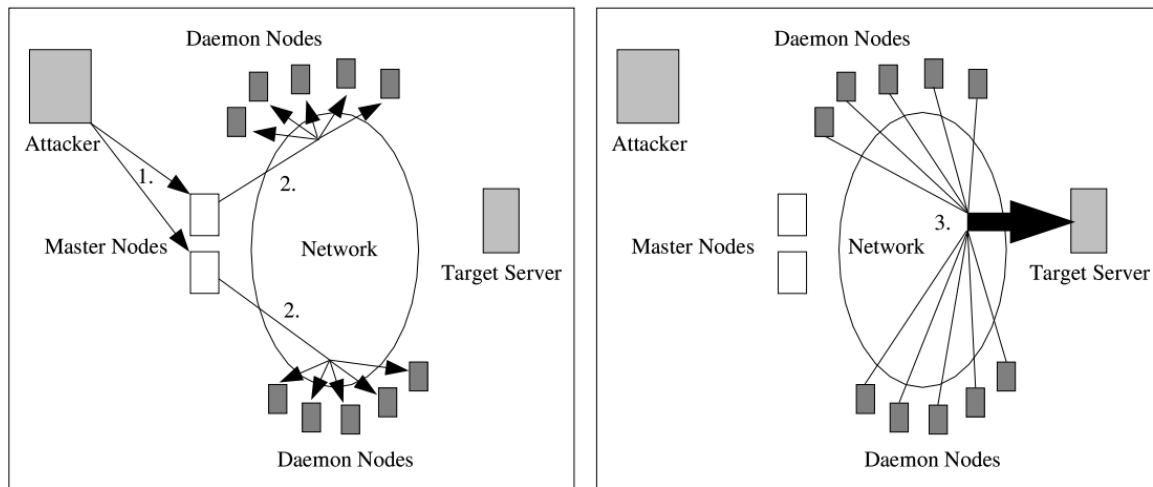
**Figure 3.1:** DDoS attack. [23]

### 3.2.2 Vulnerability scans

**Network scans** are information gathering attacks. They do not cause any damage by themselves but usually serve the purpose to gather information about a system that could be used in further attacks. Network traffic sniffing or port scans are examples of network scans. [13]

### 3.2.3 Buffer overflows

Buffer overflows are one of the most used methods of attacking a computer or network. They are used to exploit flawed programming. They work by trying to overflow buffers within the program. When this happens the overflowed data will overflow into adjacent memory. This can cause data corruption or alter the execution of a program. There are stack-based and heap-based buffer overflows. Stack-based overflows try to overflow the program stack and heap-based overflows try to overflow dynamically created memory in a program. [22]

### 3.2.4 Brute-force attacks

Brute-force attacks are a method that can be used to gain access to a target computer. This can be done by for example attempting all different kind of passwords. These attacks will eventually succeed but they can take a long time to execute. [22]

### 3.2.5 Physical attacks

Physical attacks are attacks that try to physically damage computers. These attacks can be very basic, such as cutting a wire. However, they can also be energy attacks, such as electro-magnetic pulses (EMP's) or high and low energy radio frequency (HERF and LERF) attacks. [23]

### 3.2.6 Man in the Middle

A man in the middle attack (MITM) is an attack where the attacker tries to secretly relay and possibly alter communication between two computers or networks as can be seen in Figure 3.2.
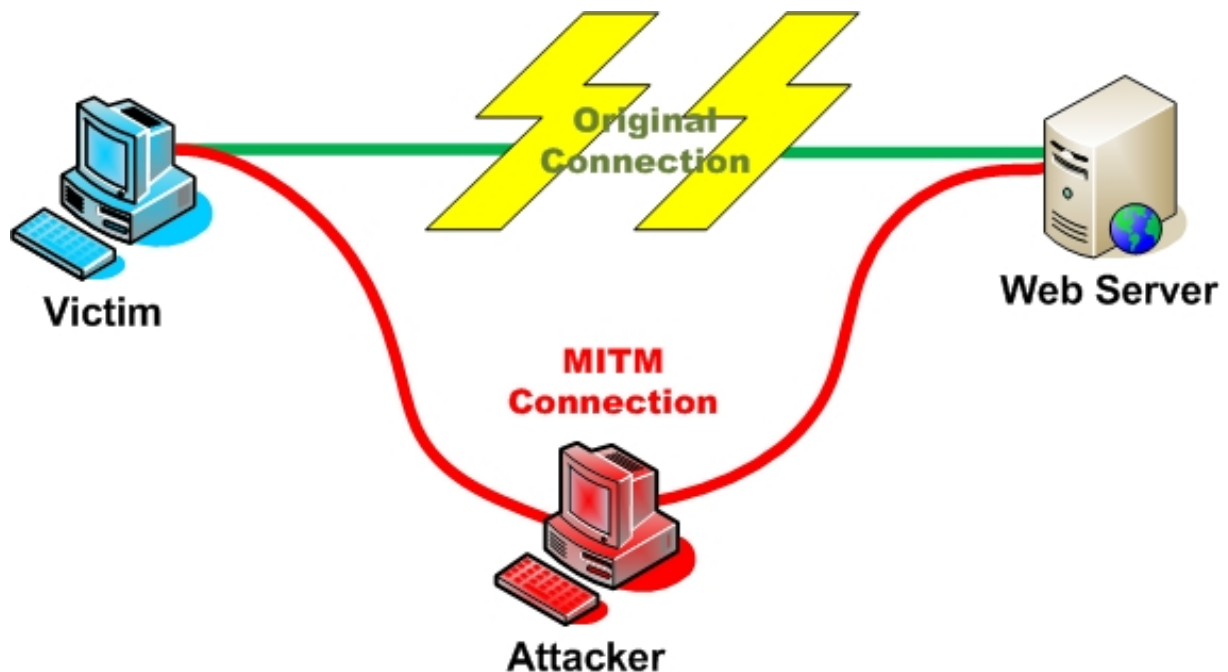
**Figure 3.2:** Man in the middle attack. [24]

## 3.3 Internal abnormal behaviour

Internal abnormal behaviour can be called malware. There are several types of malware. There are four distinct categories of malware. There are **botnets**, **viruses**, **trojan horses** and **worms**. Malware are actual programs that infect a system to execute a specific task. The task of the malware defines which category the malware belongs in.

### 3.3.1 Trojan horses

**Trojan horses** are programs disguised as harmless applications but contain malicious code. They get their name from the battle of Troy. In stories it is said that the Greeks created a giant wooden horse and filled it with soldiers. The Trojans thought the horse was a gift of surrender. They brought the horse inside Troy. At night, the Greeks came out the horse, opened the gates of Troy and defeated Troy. This is also the way a Trojan horse works. Some of the capabilities of a Trojan horse include keystroke logging, program installation and file transfers. [23]

### 3.3.2 Viruses

**Viruses** are self-replicating programs that infect and spread through other programs. An infection means that they embed themselves within another program. When that program is run, the virus also activates and runs. [23]

### 3.3.3 Worms

**Worms** are programs that replicate themselves among a network. They can spread extremely fast. Unlike viruses, they do not require infected files to spread. [23]

### 3.3.4 Botnets

**Botnets** is malware that causes infected computers to become "slaves" to the master. An infected computer is controlled externally by the bot-master without the knowledge of the owner of the infected computer. An example of this is shown in Figure 3.3. The bot-master can use the distributed network of "slave" computer to perform other malicious tasks, such as performing an DDOS attack. [13]
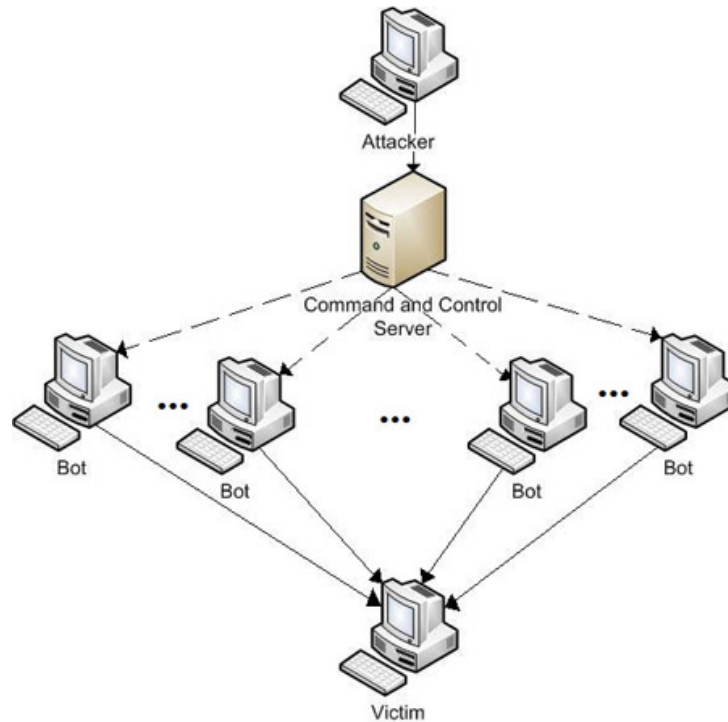


**Figure 3.3:** Botnet infection. [25]

## 3.4 Detection

An NIDS only monitors the network. As such not every attack can be detected by an NIDS. Only the attacks that actually use the network can be detected. Flow-based IDS have the additional constraint that they can only use flow data. This further limits the attacks that can be detected. The attacks that can generally be detected using a flow-based network intrusion detection systems are:

- DDOS

- Vulnerability scans

- Worms

- Botnets

Other attacks either do not use network communication, or they are not visible within the header information of network traffic. In order to detect other attacks, including **Viruses**, **Trojan horses**, **Man in the Middle** and **Buffer overflows**, other detection systems such as HIDS or Packet-based NIDS should be used. **Physical attacks** are even more difficult to detect.

However there are some special circumstances in which other types of abnormal behaviour can be detected. When more information about a network and the abnormal behaviour is known, other types of

abnormal behaviour can be detected. For example. when it is known that viruses are always distributed from the same source, it is possible to detect this distribution.This is analogous for other attacks.

Other activities such as phoning home which a keystroke logging Trojan might do, could also be detected. **Brute-force attacks** can also be detected since they might generate a lot of network traffic. However, unless the **Brute-force attack** is also a DDoS attack, the traffic mostly looks harmless from a flow-based approach. [22]

### 3.4.1 Distributed Denial of Service

A distributed denial of service can be detected by the amount of data that is being received. However, there are many different types of DDoS attacks. There are ICMP floods, SYN floods, etc. These attacks can be described in terms of traffic patterns. A traffic pattern is expressed in a couple features. These features include the number of flows and packets, the packet size, and the total bandwidth used during the traffic. For example UDP flooding can be characterised by a traffic pattern which contains a lot of packets. These patterns can be searched for during the detection phase. [26]

### 3.4.2 Network scans

There are three categories of network scans.

- Horizontal scans: a single port is scanned across many different devices.

- Vertical scan: several different ports are scanned on a single device

- Block scan: a combination of both a vertical and a horizontal scan.

Scans can also be described using traffic patterns. They are characterised with a high number of flow and a low number of packets. These can again be used to detect whether a vertical or horizontal scan occurs. [13] [26]

### 3.4.3 Worms

Worms exhibit different behaviour depending on their current state. There are two different states, a target discovery state and a transfer state. In the target discovery state, the worm explores the network to find vulnerabilities and a host to infect. During the transfer state, the worm actually transfers itself to the targeted host. The Sapphire/Slammer worm is an example of this type of behaviour. [27]

Since transfering of the worm itself happens within the payload data, a flow-based NIDS cannot detect this state. The target discovery state can be detected. Worms use techniques similar to network scans in order to find vulnerable hosts. So similar detection techniques can be used to detect worms. [28]

### 3.4.4 Botnets

Botnets usually consist out of a huge amount of infected slaves controlled by a central bot-master. Locating the individual infected slaves and isolating them is a difficult problem but is also insignificant due to the huge amount of remaining slaves. Detecting the bot-master and isolating that device is key to taking down a botnet. However indentifing botnet behaviour is a far more difficult problem than detection other types of malicious activities. [29] Malicious behaviour alone is not enough to detect botnets.

Botnets often use IRC channels in order to communicate between slaves and the bot-master. These can be indentified using flows since they often use specific ports. It is possible to use a method that does not require specific port numbers. This requires flows including extra information such as the number of packets for which the PUSH flag is set.

# Chapter 4

# Machine learning

## 4.1 What is machine learning

Machine learning is a subfield of Computer Science. It is a type of Artificial Intelligence which allows programs to learn and find patterns within data. [30]

Machine learning explores algorithms that can learn from and make predictions on data. These algorithms are called machine learning algorithms. A machine learning algorithm has to learn before it can be used to make predictions on data. **Learning** means that the algorithm has to be shown several examples of data and what the correct predictions for these examples would be. The amount of examples that have to be shown to the algorithm can be within the range of several thousands.

Once the machine learning algorithm has learned from the data, it can be used to make predictions on other data. For example, machine learning can be used in order to watch the heartrate of patients at a hospital. During the learning phase, the machine learning algorithm is shown the heartrate of a patient and the current time. After learning is done, the machine learning algorithm can predict what the heartrate of that patient should be based on the current time. This can be used to determine whether the patients heartrate is normal by comparing the predicted heartrate and the real heartrate.

There are two classes of machine learning algorithms. There is **supervised** learning and **unsupervised** learning. Supervised learning is trained using labeled data. Labeled data is data which consists of input data and the corresponding output data. Unsupervised learning uses unlabeled data. The data used to train machine learning algorithms is called a **training set**.

Supervised learning is the type of learning when the data presented to the learning algorithm is labeled. This means that the user of the learning algorithm can already make distinctions within the presented data. This is useful for regression or classification problems. Regression is explained in Section 4.2 and Classification is explained in Section 4.3.

Unsupervised learning is the type of learning when the data presented to the learning algorithm is unlabeled. The learning algorithm has to find structure in the given data.

A **training sample** is a data point in an available training set that is used in a predictive modeling task. For example, if machine learning is applied to spam filtering, the training set is a collection of emails, some of which are spam emails. A training sample would be a single email. Alternative names are a training example or training instance.

Machine learning is applied for predictive modeling. In predictive modeling, a particular process or event is modeled. Using a training set, the model tries to learn or approximate a particular function that, for example, lets us distinguish spam from non-spam email. This function is called the target function. The function the model makes to approximate the target function is called the **hypothesis**.

To model the particular process or event, one or more **features** are extracted. A feature is an individual property. All features combined define the model of the particular process or event. In the example of mails, a feature could be the textbody, or it could be the sender of the email. It could even be individual characters which appear in the email. Which features are chosen is completely dependent on which problem is being solved.

This chapter gives an introduction to the mathematical background of supervised machine learning. It starts with lineair regression which is a simple category of machine learning algorithms. The principles behind lineair regression are the same as for other machine learning algorithms. Afterwards logistic regression is used to explain how classification works within machine learning. It begins by explaining what exactly the hypothesis is and how it can be constructed. [31]

## 4.2   Lineair Regression

Lineair regression is a statistical approach to model the relationship between an "output" value $y$ and one or more "input" values $x_1...x_n$. The "input" values are called features. It belongs to the category of supervised learning. An example of this can be seen in Figure 4.1. The black dots represent the data to be modeled. The blue line is the model. This example only has one "input" value. This specific case of regression is called simple linear regression. [32].

### 4.2.1   Hypothesis

Machine learning relies heavily on a hypothesis which is given the notation $H_0(x)$ in equations. This is a function that transforms a given input to the machine learning algorithm into the required output. It is a function that tries to model the target function, it tries to give a function which fits the input data. The input are features, denoted by $x$. The values for $\theta$ are unknown values. These values are chosen during the training of the machine learning algorithm and determine how accurate the algorithm is. When only one feature is considered, a hypothesis is a function of the form [33]:

$$H_0(x) = \theta_0 + \theta_1 * x \tag{4.1}$$

For example, we could use the grades of high school students to predict their chance of success at university. This is shown in Figure 4.1. The $x - axis$ represents the grades (on a scale from 0 to 10) for students and the $y - axis$ is the chance of success. Given is input data (the black points) and from that data a hypothesis (the blue line) is constructed.

The hypothesis function can be generalised for $N$ features. It generally has the form:

$$H_0(x) = \theta_0 x_0 + \theta_1 x_1 + ... + \theta_n x_n \tag{4.2}$$

$x_0$ always has the value 1. The $\theta$ values and the $x$ values can be represented using vectors. Now the hypothesis function can be written in a more convenient way:

$$H_0(x) = \theta^T X = [\theta_1, \theta_2, ..., \theta_n]^T [x_0, x_1, ..., x_n] \tag{4.3}$$
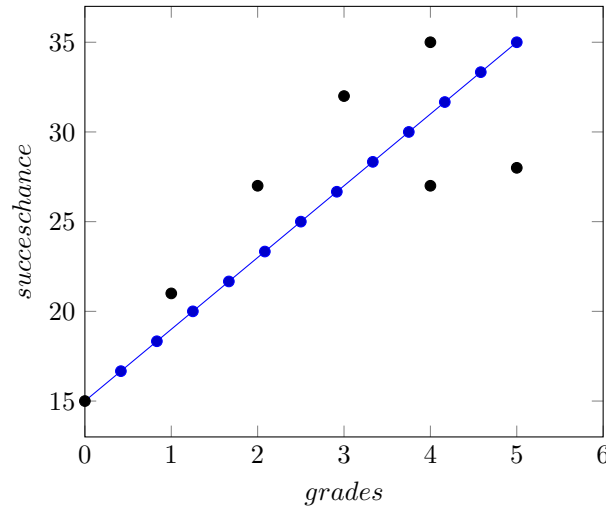
**Figure 4.1:** Lineair Regression

The hypothesis is the core of a machine learning algorithm. During the training process, the algorithm learns what the hypothesis looks like. More specifically, it tries to find correct values for the vector $\theta$ (as seen in equation 4.3. Once the hypothesis has been determined, the difficult work is done. The hypothesis forms the predictive model. The prediction itself is straightforward. A new datapoint is entered into the hypothesis equation and it returns the predicted value.

### 4.2.2 Cost function

In order to construct a good hypothesis function, good values of $\theta$ have to be chosen. This can be seen as a minimization problem. The difference between any output $y$ and $H_0(x)$ has to be minimized. More concretely, the squared difference has to be minimized. This is the MSE, "Minimum Squared Error" function or also called the cost function. The notation $x^{(i)}$ and $y^{(i)}$ is to denote the $i$th training sample. With dataset size $m$, the MSE is [33]:

$$J(\theta) = \frac{\sum\limits_{i=1}^{m}(H_0(x^{(i)}) - y^{(i)})^2}{2m} \tag{4.4}$$

Seeing this equation already makes it clear that the number of training samples is important. From statistics, it is known that the population of data points, in this case training samples, have to be large enough. This problem is addressed later in Section 4.4.

### 4.2.3 Gradient descent

To solve the minimization problem, the first step is to start with $\theta$ and keep changing the values to minimize $J(\theta)$. This is an iterative approach. Gradient descent is such an algorithm that can be used to find a solution to the minimization problem. Gradient descent is an algorithm that uses the gradient or derivative of a function to find a local minimum of that function.

In order to easily explain the gradient descent algorithm, a analogy to a ball rolling down a hill can be made. [34] Let's say we have a landscape such as in Figure 4.2. The horizontal location of the ball represents the values of $\theta$, the height of the ball represents the MSE. We want to get the ball to the lowest possible point, in order to minimize the MSE. To do this, the ball rolls down the hill until it comes to rest at the lowest point. Every iteration of gradient descent will bring the ball closer to the lowest point.

Of course, when a ball rolls down a hill, it does not just roll down and immediately stop. The ball takes a path similar as described in Figure 4.2. When the ball finally comes to rest, it can be said that the algorithm converges to that horizontal position.
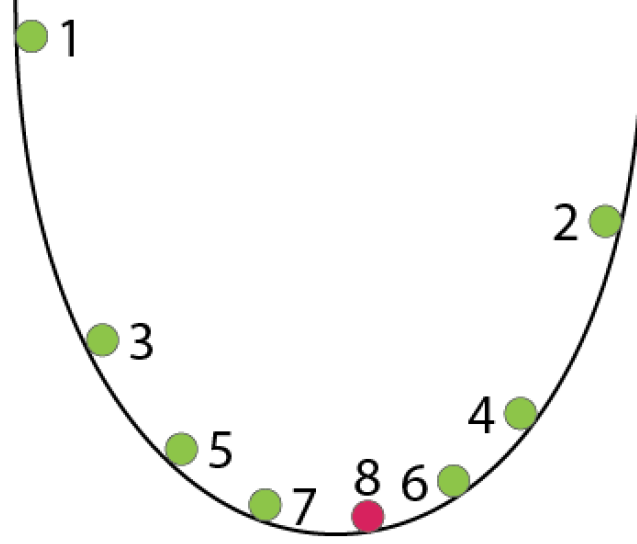


**Figure 4.2:** Iterations of gradient descent. [35]

A gradient descent algorithm does this using the following algorithm with a simultanious update for all values of $\theta$ [33]:

repeat until convergence {

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \tag{4.5}$$

}

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\sum\limits_{i=1}^{m} (H_0(x^{(i)}) - y^{(i)}) * x_j^{(i)}}{m} \tag{4.6}$$

The equation uses the derivative. A derivative indicates the rate of change, in this case, how steep the hill is. If the hill is very steep, then we move the ball a lot more than when the ground beneath the ball is almost flat.

$\alpha$ is the learning rate of the gradient descent. The value of $\alpha$ describes how fast the gradient descent algorithm approaches the local mimimum. If $\alpha$ is too small, the gradient descent can be very slow. In the other case, if $\alpha$ is too large, the gradient descent can overshoot the local mimimum. It may fail to converge and could even diverge.

To go back to the example of the rolling ball, having $\alpha$ too large, could be seen as having the ball roll down the hill extremely fast. Because of this, the ball could come to rest at another point in the land-scape, it might have flown over the hill.

The value of $\alpha$ does not need to change during the gradient descent, since the closer the gradient descent gets to the local mimimum, the smaller the derivative becomes, and smaller steps will be taken. If $\theta_j$ is already a local minimum, the derivative is $0$ and the gradient descent will not change the value of $\theta_j$. [36]

### 4.2.4   Normal equation

There is another method that could be used in place of gradient descent, a normal equation. This is a method to solve for $\theta$ analytically. But this method becomes slow when there are a lot of features. $X$ is a $m$ X $n$ matrix constructed by putting all training samples (vectors of features) together. The notation $x^{(i)}$ and $y^{(i)}$ is used to denote the $i$th training sample. With dataset size is denoted by $m$.

$$\theta = (X^T X)^{-1} X^T y \tag{4.7}$$

But what happens when $(X^T X)$ is non-invertible, or singular? This means there are redundant features or more features than training samples. There are mathematical models, such as the pseudo-inverse to still compute a correct result. There are still other methods that could replace gradient descent, such as conjugate gradient, BFGS and L-BFGS. Thinking about matrices and non-invertibility can help to be able to understand how features relate to eachother.

### 4.2.5   Feature scaling

The main idea behind feature scaling is to make sure that the different features are on a similar scale. The reason behind this is to optimize the gradient descent algorithm. The gradient descent will converge much more quickly with feature scaling. With feature scaling, the partial derivative will be larger and because the features are in scale, the values of $\theta$ will also be similar in scale.

When one feature is very large and another is very small, the values of $\theta$ need to be small for the first feature and larger for the second, since the cost function needs to be minimized. This means that the values of $\theta$ need to change more and thus, requires more iterations for the gradient descent algorithm. The range that should approximately be used is $-1 < x < 1$.

Mean normalization could be used. This replaces each $x_i$ with $\dfrac{x_i - \mu_i}{s_i}$, where $\mu_i$ is the average value of all $x_i$ values and $s_i$ is the standard deviation.

Lets use a numeric example. There are two data points, $(0.5, 3, 1)$ and $(0.7, 4, 2)$ with $(x_1, x_2, y)$. The hypothesis is:

$$H_0(x) = \theta_1 * x_1 + \theta_2 * x_2 \tag{4.8}$$

In Figure 4.3, it can be seen that the values of $\theta$ quite quickly fall into the blue zone, where the cost function is minimal. The values of $\theta$ also seem to be equally large when the cost function is minimal. This in contrary to Figure 4.4. Here, both values of $\theta$ need to change a lot. They are not even remotely close to eachother.
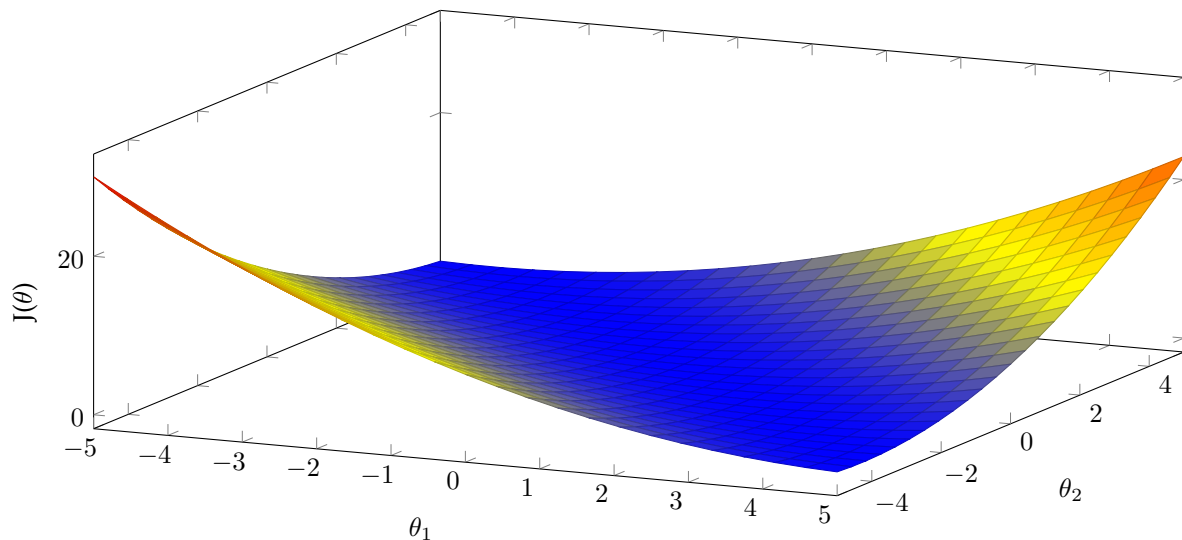
**Figure 4.3:** Cost function with feature scaling

When $\theta_2$ is near $-4$, the gradient descrent will change $\theta_1$ towards a larger value. However, when $\theta_2$ is near $4$, the gradient descrent will change $\theta_1$ towards a smaller value. In the gradient descent algorithm, $\theta_2$ will converge towards approximatly $-1$ and the actual value of $\theta_2$ will change between being larger and smaller than this value. Because of this, the value of $\theta_1$ takes a longer time to converge.
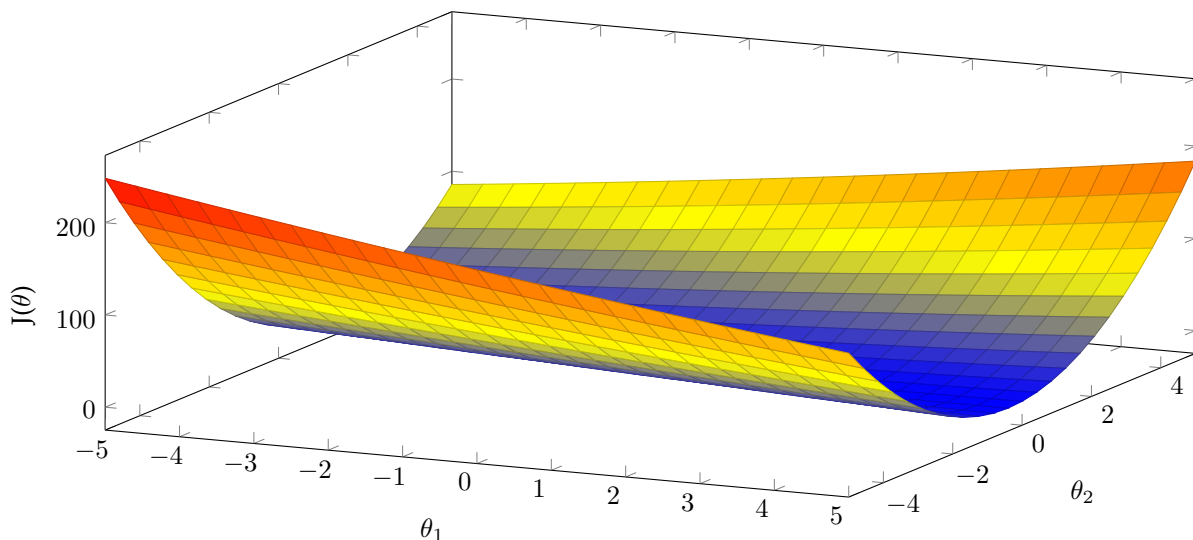


**Figure 4.4:** Cost function without feature scaling

## 4.3   Classification with logistic regression

In a classification model, the machine learning algorithm tries to sort data into different classes. The simplest version is binary classification. For example, "is the email spam or not". In lineair regression, the hypothesis can output values other than the classes that exist. However there is a method, logistic regression, which constrains the hypothesis to the available classes.
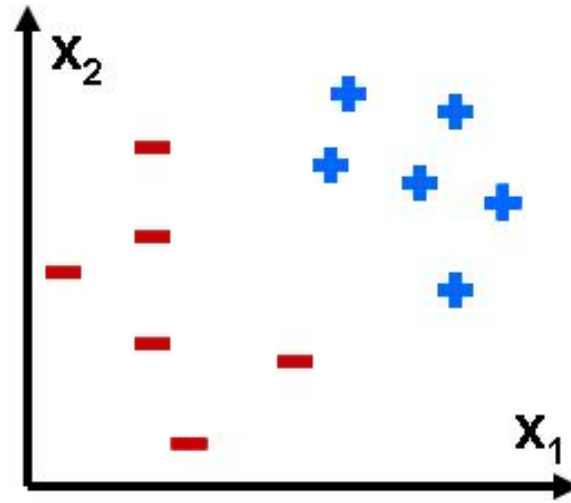
**Figure 4.5:** Binary classification. [37]

Figure 4.5 shows a simple binary classification example. $x_1$ and $x_2$ are features. Some points belong to the minus class, others belong to the plus class. A logistic regression or classification algorithm will learn from training data what data from the minus class and data from the plus class looks like. Afterwards, the classifier can be used to predict whether an input sample belongs to the minus class or the plus class.

### 4.3.1 Logistic regression

In logistic regression, a hypothesis is needed which outputs whether an input datapoint belongs to a class or not. It can also be explained as, "what is the chance that the input belongs to a class"? Now the hypothesis needs to output a percentage, the chance that the input belongs to a certain class.

Then a decision boundary is used. This decides whether the input belongs to the class or not. Usually the decision boundary is $0.5$. If the hypothesis outputs a value higher than the decision boundary, it can be said that the input belongs to the class for which it is being tested. Logistic regresion uses the Sigmoid or logistic function for the hypothesis:

$$g(z) = \frac{1}{1 + e^{-z}} \tag{4.9}$$
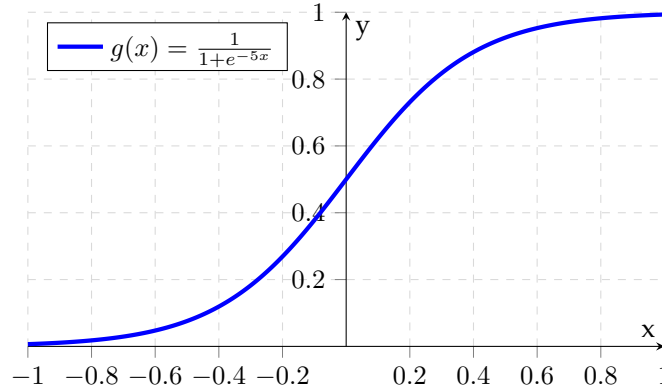
**Figure 4.6:** Sigmoid function

Using this function, the hypothesis can be written as:

$$H_0(x) = g(\theta^T X) = \frac{1}{1 + e^{\theta^T X}} \tag{4.10}$$

### 4.3.2   Cost function

The cost function from lineair regression cannot simply be applied to logistic regression. For classification it is important that data that belongs to the class has an output closer to 1, and that data that does not belong to the class has an output closer to 0. To do this a different cost function should be used. The notation $x^{(i)}$ and $y^{(i)}$ is used to denote the $i$th training sample. With dataset size is denoted by $m$. A helper function $Cost$ is used. This helps to see the parallel with the cost function for linear regression, as seen in Section 4.2.2. [38]

$$J(\theta) = \frac{\sum\limits_{i=1}^{m} (Cost(H_\theta(x^{(i)}), y^{(i)}))}{m} \tag{4.11}$$

$$Cost(H_\theta(x^{(i)}), y^{(i)}) = \begin{cases} -log(H_\theta(x^{(i)})), & \text{if } y^{(i)} = 1 \\ -log(1 - H_\theta(x^{(i)})), & \text{if } y^{(i)} = 0 \end{cases} \tag{4.12}$$

The given cost function accomplishes exactly what is wanted. If the expected output is 1 (belongs to a class), then the cost becomes greater as the output goes to 0 and analogous if the expected output is 0. Gradient descent for logistic regression is exactly the same as for lineair regression except with a different hypothesis.

A very simple example of the cost function would be the following. Take one data point, (3) with outputs 1. The hypothesis seen in Figure 4.7 and is:

$$H_0(x) = \frac{1}{1 + \exp \theta_1 * x_1} \tag{4.13}$$

The cost function is: $-log(\theta_1 * 3)$ which is graphed in Figure 4.8 becomes larger when the hypothesis goes towards 0, and goes to 0 when the hypothesis goes towards 1. This is indeed the behaviour that is expected from the cost function for logistic regression.
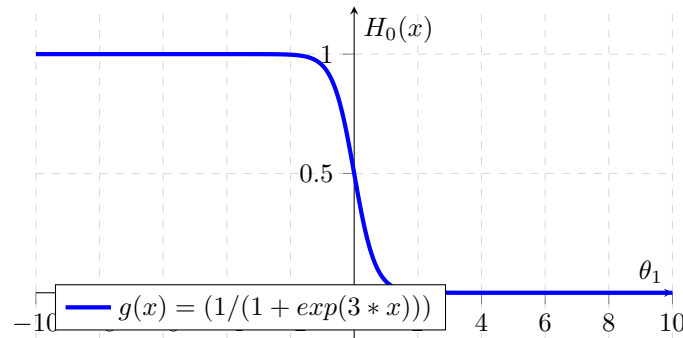
**Figure 4.7:** Hypothesis function example
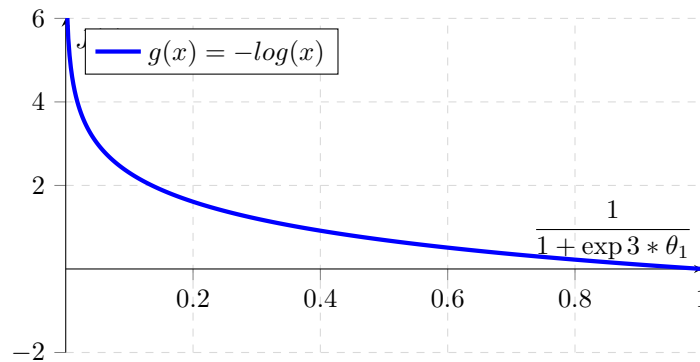


**Figure 4.8:** Cost function example

### 4.3.3 Multi-class classification

The above hypothesis and cost function are for binary classification. To compute multi-class classification, the One-vs-all algorithm could be used. This algortihm splits the multiples classes into two groups. One group contains one class, the other group contains all other classes. Using these two new groups, the algorithms for binary classification can be used. In other words, for each class $i$ a different logistic regression classifier $H_\theta(x)$ is trained to predict the probability that $y = 1$. On an input x, the most probable class is chosen.

Another method called One-vs-One could be used. This method compares all pairs of classes. It checks, "does the input belong rather to class A as compared to class B". It does this for all pairs and decides which class most likely contains the input.

The One-vs-One method uses $n_c lasses * (n_c lasses - 1)/2$ different logistic regression classifier $H_\theta(x)$. Each of these classifiers is trained to fit a pair of classes. [39]

To provide an example, Figure 4.9 can be used. On the left, binary classification is shown. The hypothesis and cost function as seen in the above sections can be used to do the classification. On the right, multi-class classification is used. There are three classes: triangles, squares and crosses.
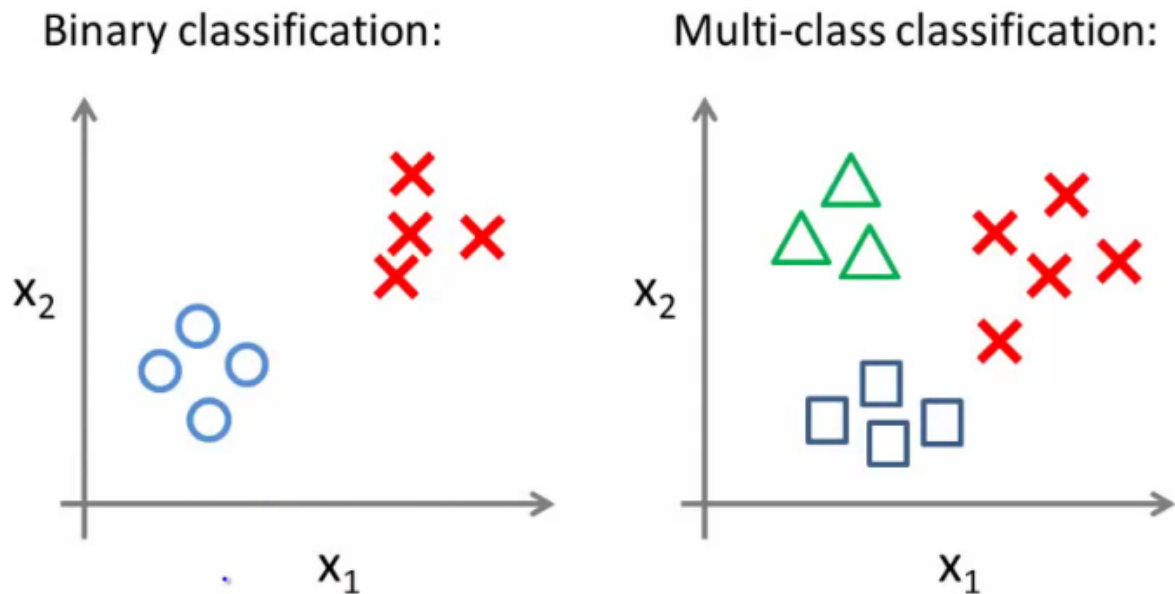
**Figure 4.9:** On the left, binary classification is shown. On the right, three different classes are used.
[40]

One-vs-all classification will have to use three classifiers to predict which class a given input sample belongs to. The first classifier predicts whether the input samples belongs to the triangles or to another class. The second classifier predicts whether the input samples belongs to the squares or to another class and analogous for the third classifier. This method can quite quickly find the result. If the first classifier is used first and it is found that the input sample belongs to the triangles class, the prediction is done. [40]

One-vs-One classification will have to use six classifiers. Each classifier is trained to predict whether an input sample belongs to the triangles or the squares, to the triangles or the crosses, and so on. If the classifiers predict that the input samples belongs rather to the triangle class than to the square class and rather to the triangle class than to the crosses class, it can be predicted that the input sample belongs to the triangle class. Because the One-vs-One classification has to use more classifiers than One-vs-All, it is also slower. However, because it compares all pairs it is also more accurate. [40]

## 4.4  Overfitting

When the data is not modeled correctly and the model is too precise for the training data, a problem called overfitting occurs. Models that are overfitted have high variance, there are too many possible hypotheses. In other words, if there are too many features, the hypothesis may fit the training set very well but fails too correctly predict new examples. The hypothesis becomes very complex. An example of this can be seen in Figure 4.10. In a similar way, underfitting may occur.
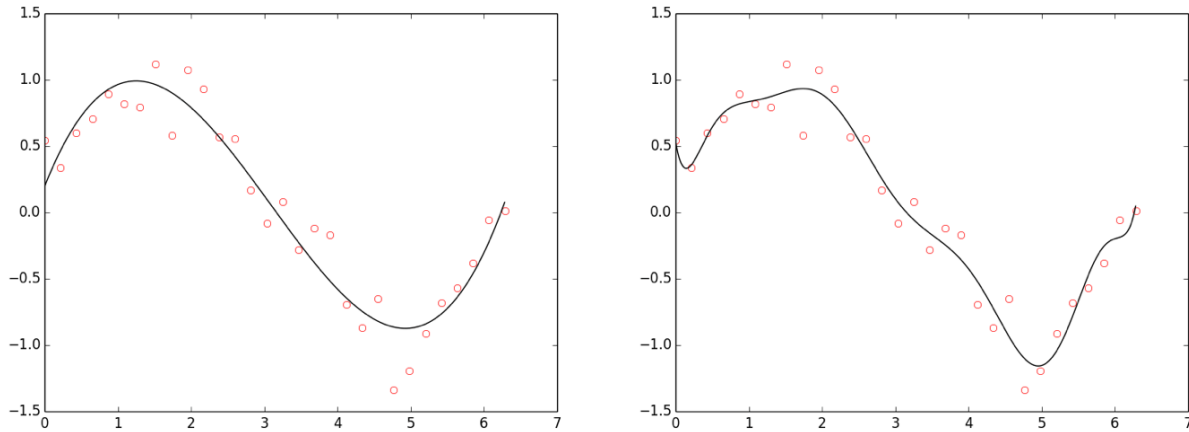
**Figure 4.10:** The points are generated by a sin function with Gaussian noise. Left: good fit (polynomial of degree 3), Right: overfit (polynomial of degree 10). [41]

There are methods to avoid overfitting. It is possible to reduce the amount of features. This can be done manually or by using a model selection algorithm which is explained in Section 7.2.3. Another option is regularisation. This method keeps all features but manages the values of the parameters of $\theta$. Regularisation works well when there are a lot of features that all contribute to be able to make correct predictions $y$.

A hypothesis is complex when the values of $\theta$ are too large. Regularisation tries to solve this problem. When regularisation is used, the chosen values of $\theta$ become lower. For lineair regression, this can be done in the cost function by redefining the cost function as:

$$J(\theta) = \frac{\sum\limits_{i=1}^{m}(H_0(x^{(i)}) - y^{(i)})^2 + \lambda * \sum\limits_{j=1}^{m}(\theta_j^2)}{2m} \tag{4.14}$$

The only change to the cost function is that a penalty is added for large values of $\theta$. When values of $\theta$ become larger, the cost function also becomes larger. When this cost function is minimized lower values for $\theta$ will be chosen. Only $\theta_1$ till $\theta_m$ should be regularised. $\lambda$ is called the regularisation parameter. Because of this small modification, the hypothesis that is constructed will be simpler.

Lets use a numeric example. There are two data points, $(3, 1)$ and $(5, 2)$. The hypothesis is:

$$H_0(x) = \theta x_1 \tag{4.15}$$
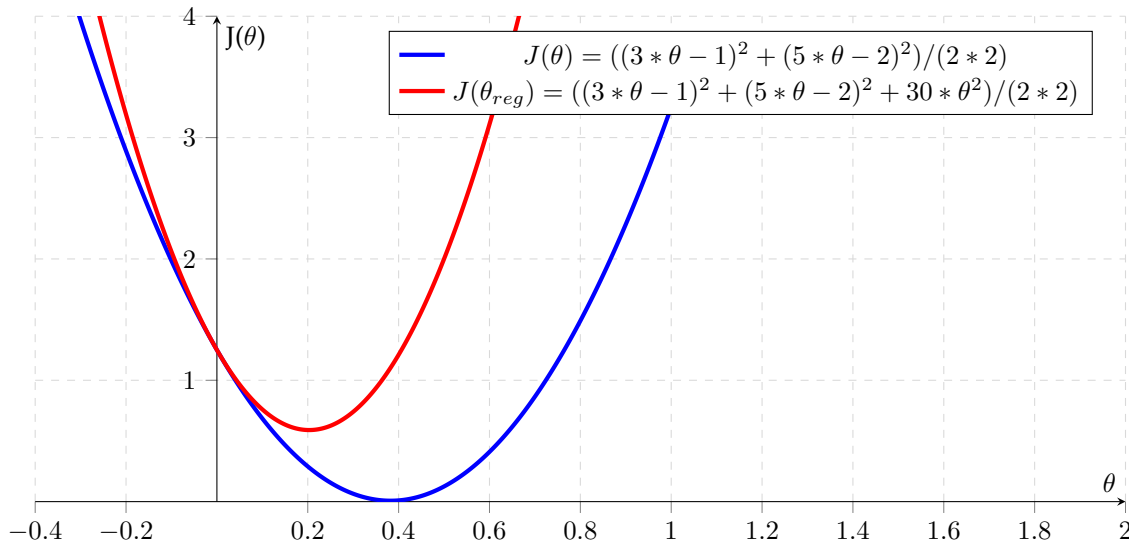
**Figure 4.11:** Effect of regularisation

In this example, a cost function is constructed with and without regularisation. The blue graph shows the cost function without regularisation, the red graph shows the cost function with regularisation. It can be seen that having regularisation does change the value of $\theta$ where the cost function is minimal, which is exactly what is intended to happen with regularisation.

## 4.5   Distance metrics

Machine learning algorithm also use distance metrics. These can be used to define how to compute distances between different data points that are fed to a machine learning algorithm. There are several different methods to compute distances, both for continuous features and discrete features.

**The Euclidean distance** is the distance between two points in the Euclidean space. This can be seen in Figure 4.12. With features $x = (x_1, x_2, ..., x_n)$ and $y = (y_1, y_2, ..., y_n)$, the Euclidean distance is defined as $\sqrt{\sum_{i=1}^{n} ||x_i - y_i||^2}$. When features are continuous, the Euclidean distance is a good metric. However, discrete features cannot be handled by the Euclidean distance. [42]

For example, when a feature represents something such as "country of origin". Countries could be considered continuous. For example the distance between countries could be calculated using $(latitude, longitude)$. In this example, the feature represents the "country of birth" of a person. In that case it does not make sense to use a continuous feature. In this case, the "country of birth" is either equal or not equal. This also shows that a single feature such as a country can be represented as either continuous or discrete depending on the problem that is being solved.

Features are almost always represented in numeral methods. "Belgium" could be 1, "Netherlands" could be 2, "France" could be 3, and so on. The difference between "Belgium" and "Netherlands" should be the same as "Belgium" and "France".

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$
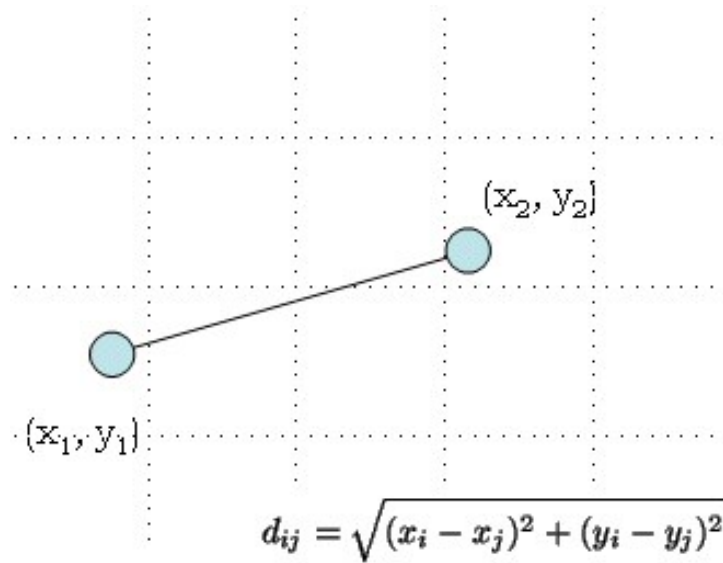
**Figure 4.12:** Euclidean Distance. [43]

**The Hamming distance** is defined as the number of positions at which the corresponding symbols are different.[44] For example, "John" and "Jack" have a Hamming distance of 3. "Jack" and "Jace" have a Hamming distance of 1. This can be seen in Figure 4.13 Going back to the example of countries. Each country can be represented as a number. Each number is seen as a different symbol. For example 10 is seen as a single number. With the countries, the Hamming distance can only be 0 (countries are the same) or 1 (countries are different).
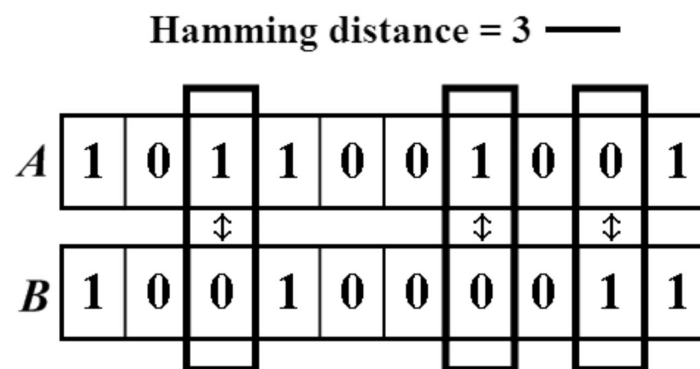


**Figure 4.13:** Hamming Distance. [44]

**The Chebyshev distance** is a distance which is defined as the largest difference between the features. With features $x = (x_1, x_2, ..., x_n)$ and $y = (y_1, y_2, ..., y_n)$, the Chebyshev distance is defined as $\max_{i=1}(||x_i - y_i||)$. It is also known as chessboard distance. [45] It is the amount of moves a king needs to move to another space on a chessboard as seen in Figure 4.14.

**Figure 4.14:** Chebyshev Distance. [46]

**The Manhattan distance** is similar to the Euclidean distance. With features $x = (x_1, x_2, ..., x_n)$ and $y = (y_1, y_2, ..., y_n)$, the Manhattan distance is defined as $\sum_{i=1}^{n} ||x_i - y_i||$. The Manhattan distance is the sum of the difference between each feature. [47] The Manhattan distance in a two-dimensional space is the sum of the difference of the $x$-values and the difference of the $y$-values. An example can be seen in Figure 4.15.
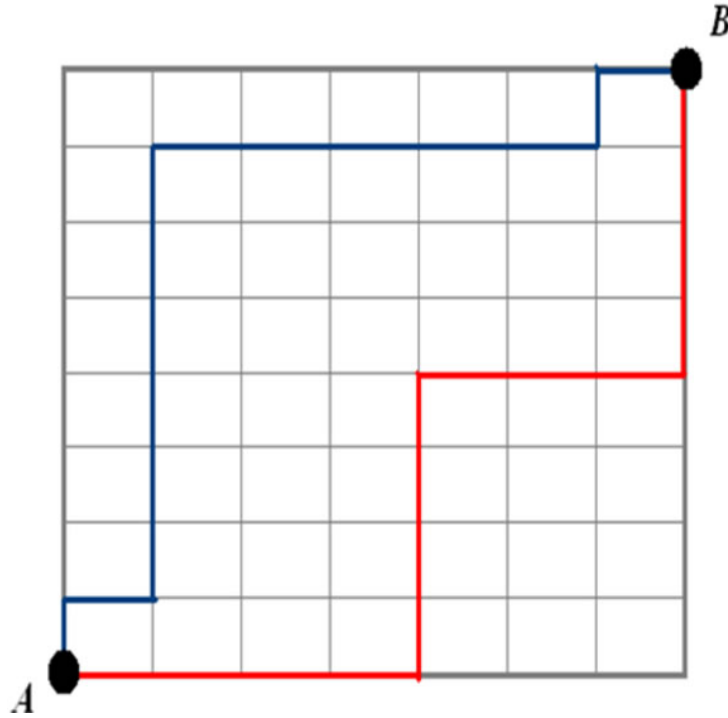
**Figure 4.15:** Manhattan Distance. [47]

**The Canberra distance** is a weighted version of the Manhattan distance. With features $x = (x_1, x_2, ..., x_n)$ and $y = (y_1, y_2, ..., y_n)$, the Canberra distance is defined as $\sum_{i=1}^{n} \frac{|x_i - y_i|}{|x_i| + |y_i|}$. The Canberra distance is the same as the Manhattan distance divided by the sum of the different values. [48]

**The Minkowski distance** is a generalisation of the Euclidean distance and the Manhattan distance. Below the formulas for the Euclidean distance and the Manhattan distance can be seen. However, they are written slightly different. These formulas are almost the same.

$$EuclideanDistance(x, y) = (\sum_{i=1}^{n} ||x_i - y_i||^2)^{\frac{1}{2}} \tag{4.16}$$

$$ManhattanDistance(x, y) = (\sum_{i=1}^{n} ||x_i - y_i||^1)^{\frac{1}{1}} \tag{4.17}$$

A generalisation can be written as: $(\sum_{i=1}^{n} ||x_i - y_i||^p)^{\frac{1}{p}}$. $p$ is a free variable which can be set. If $p$ goes to infinity, the Chebyshev distance is obtained. The effect of $p$ is shown in Figure 4.16. This shows that the Minkowski distance is very flexible and powerfull. [49]
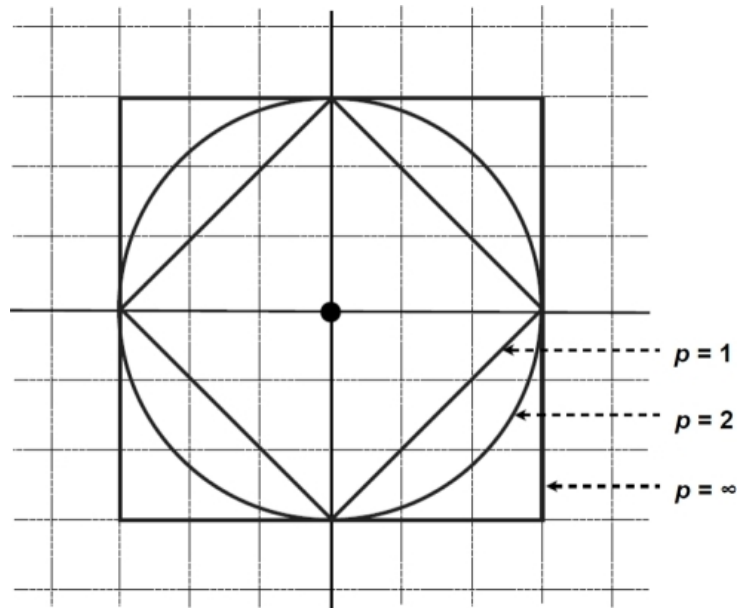
**Figure 4.16:** Minkowski Distance. [50]

The performance of each of these distances on the accuracy depends on the problem that is being solved. Finding the correct distance metric is a matter of trial and error.

## 4.6　Summary

At the core of a machine learning algorithm is the hypothesis. The hypothesis is the function that is used to fit the training data, it is used to predict values for new data. The hypothesis is constructed using the features that represent the data given to the machine learning algorithm and a set of unknown values $\theta$.

These values $\theta$ are chosen, as such that the hypothesis is the most accurate. In order to do this, a cost function is constructed. The cost function measures the difference between a known output and a predicted output, usually the MSE is used. The values for $\theta$ can be found by minimizing the cost function. This is done by using the gradient descent algorithm.

# Chapter 5

# Algorithms

This chapter gives an introduction to different machine learning algorithms. It starts with two popular supervised learning algorithms, Support Vector Machines and K-Nearest Neighbors. Afterwards it explains how neural networks work. Finally some unsupervised techniques such as clustering and anomaly detection are discussed.

## 5.1 Support Vector Machines

Support vector machines or SVMs is a supervised learning algorithm which offers an alternative view on logistic regression. Support vector machines try to find a model which divides 2 classes exactly with the same amount of margin on either side as shown in Figure 5.1. Samples on the margin are called the support vectors.
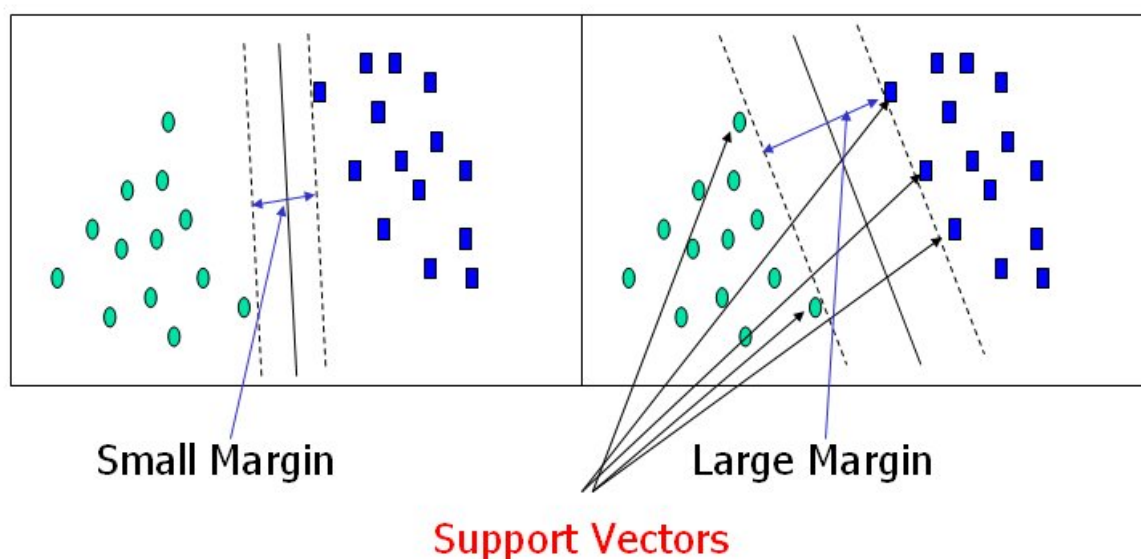


**Figure 5.1:** Support vector machines with their support vectors. [51]

### 5.1.1 Linear and non-linear classifiers

Normally Support Vector Machines, as well as general logistic regression work with linear classifiers. A classifier can be linear or non-linear. A linear classifier has a decision boundary which is a linear function. This can be visualised by visualising that there is a line drawn through the data. On one side of

this line, the data belongs to one class. Data on the other side of this line belongs to another class. [52]

Sometimes problems cannot be solved with linear classifiers. When training data contains a lot of noise for example. Noise in data could mislead the learning method and increase the classification error. Figure 5.2 shows classification of webpages being Chinese or English web pages. This seems to be a linear problem and a linear classifier has divided the space in the two classes (dashed line and short dashes). However, there are three noise documents (marked with arrows). The noise is misleading the classifier. It increases the classification error. [53]

However, when a non-linear classifier is applied, the noise is not a problem anymore. The noise elements belong to their respective classes. This classifier performs better than the linear classifier. In this example, the noise elements are considered important and that has as effect that the non-linear classifier is not considered overfitted. [53]
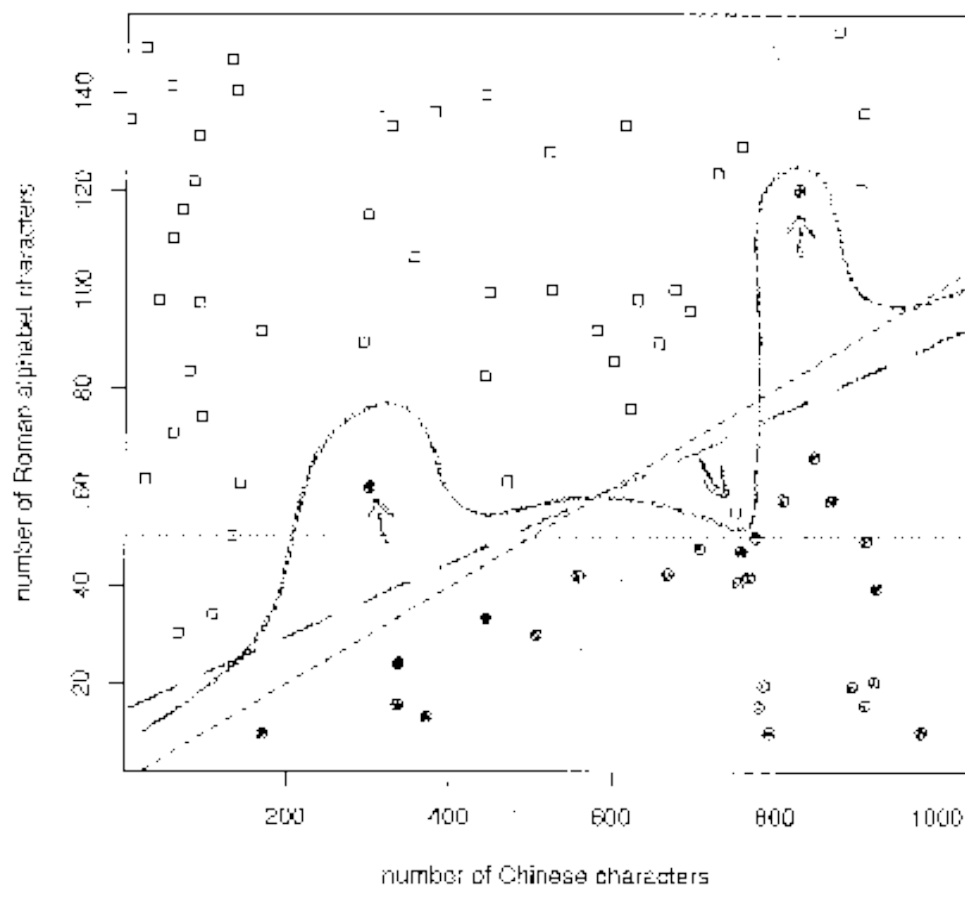


**Figure 5.2:** Linear and non-linear classifiers. [53]

## 5.1.2   Kernels

In order to adapt Support Vector Machines to be able to fit non-linear classifiers, some adjustments need to be done. This can be done with kernels. A kernel is a similarity or inverse distance function. Which can be any of the functions seen in Section 4.5. The function compares two inputs (data points) and computes their similarity. The more similar two data points are, the smaller the distance in between them. This means that there is an inverse relation between distance metrics and kernels.

Using a kernel in Support Vector Machines, rather than preprocessing the entire feature space, all features at once, is called the kernel trick. [54].

The kernel trick is often computationally cheaper than the explicit computation of "new" features which represent data and their similarity. Normally features are extracted from data and then fed into a machine learning algorithm. Kernels offer an alternative.

The kernel should be a function to compare input data. The kernel, along with labeled data is then used to construct features. In literature, when no special kernel is used, the algorithm is said to use a linear kernel. One of the most basic kernels is just using a simple distance metric. This could be the inverse of the Euclidean distance as seen in Section 4.5.

A more advanced type of kernels are called Gaussian (or RBF) kernels. Gaussian kernels are also one of the most popular type of kernels and is often used. Gaussian kernels use the squared Euclidean distance between two features. However, instead of taking the inverse of the squared Euclidean distance, it uses the exponential function. The exponent is the negative of the squared Euclidean distance.

## 5.2 K-Nearest Neighbors

The K-Nearest Neighbors or KNN algorithm is a supervised learning algorithm which computes the classification by looking at the classes of the K-Nearest neighbors of the training data. The K-Nearest Neighbors is an Instance-based algorithm. [55] The chosen class is the class most common among its K-Nearest neighbors, this can be seen in Figure 5.3. K is typically a small number. When K is equal to 1, the assigned class is the same as the class of the closest sample.
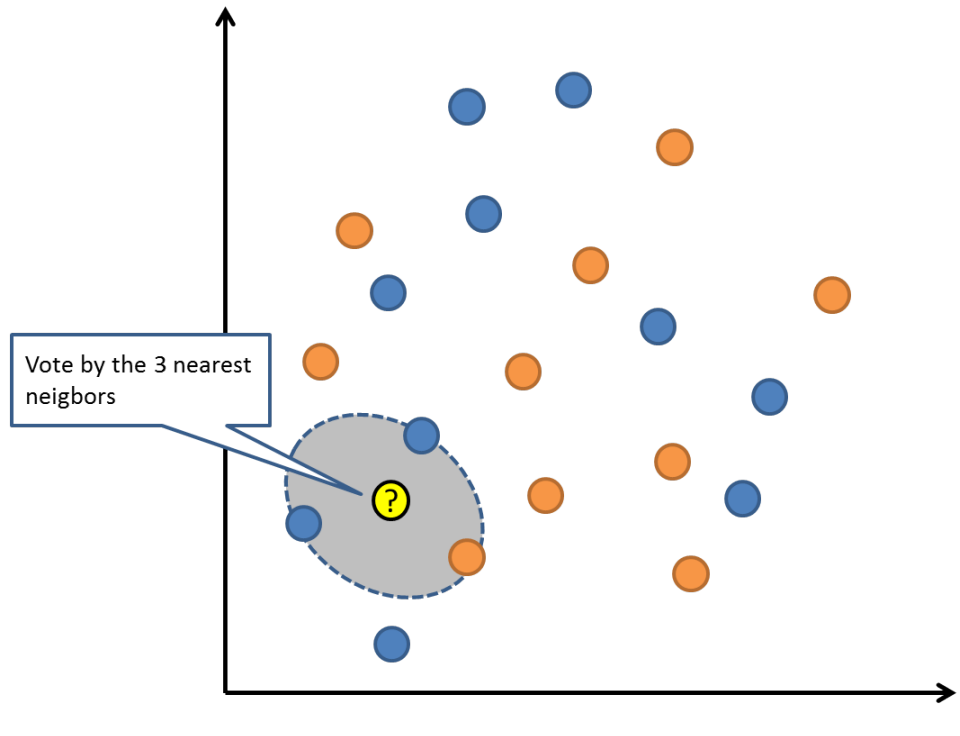


**Figure 5.3:** K-Nearest Neighbors. [56]

When training the algorithm, the input data and classes are stored. There are mutliple methods to compute the distance between data. Euclidean distance can be used for continuous data. For discrete variables another metric can be used, such as the Hamming distance. There are many different methods for computing the distance, these are explained in Section 4.5.

### 5.2.1   KNN drawbacks & solutions

A major drawback of the KNN algorithm is the weakness to skewed data. Skewed data is data that belongs to a class that is underrepresented in the complete data set. Since the class is chosen based on the most popular nearest class, these popular classes may dominate the prediction. One simple solution is to gather more data. More realistically, the problem can be overcome by taking the distance between the input data and the neighbors into account. If this does not work, it is possible to use learning algorithms to select the best data from the data set so that no class is underrepresented. These methods are far more advanced and have a higher computational cost.

In Figure 5.5, an example is shown with 3-class classification. In the this example, the weights are uniform. The score that decides which class is going to be predicted is computed from a majority vote. The example from Figure 5.6 assigns weights proportional to the inverse of the distance between the data point which is being predicted and any $k$ neighbors. The distance function used to select the class is similar to $\dfrac{1}{\text{distance between x and y}}$ with $x$ the data point for which a class is going to be predicted and $y$ a neighbor of $x$. This is the method to address skewed data.

If $k$ would be set to $2$, then the $2$ closest neighbors are chosen. The class is then chosen based on the inverse distance. Assume that the $2$ closest neighbors belong to classes $A$ and $B$. The first neighbors is closer to the data point for which a class is going to be predicted. Then the class that is chosen for the data point is $A$, not $B$. In Figure 5.4, it can be seen that when the distance (red line) increases, the weight decreases (blue line). All weights that belong to neighbors of the same class are added together, the class with the highest score is chosen.
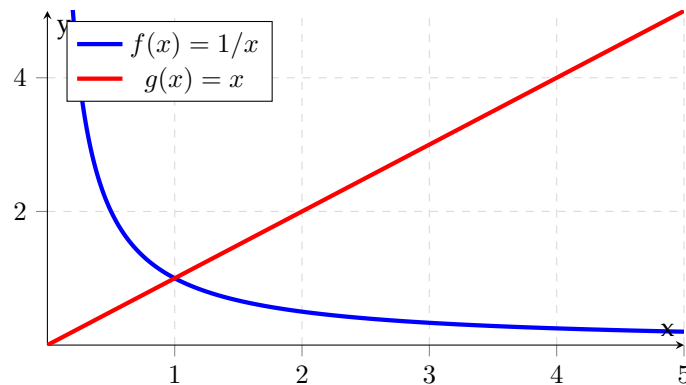


**Figure 5.4:** Inverse distance function
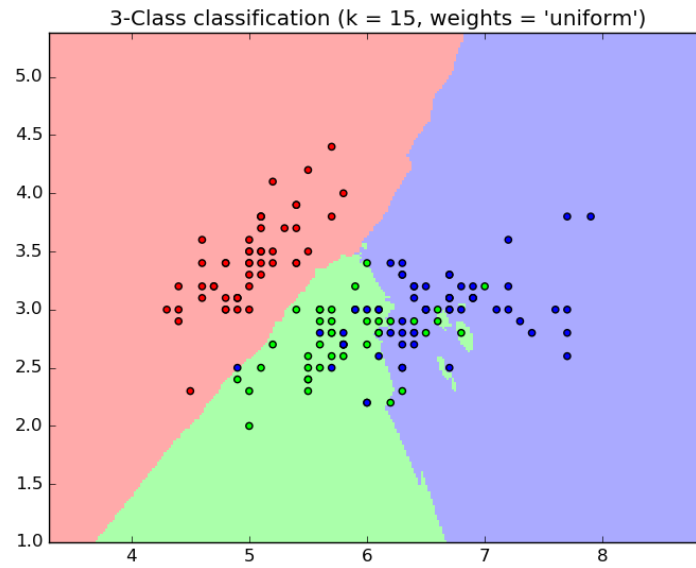
**Figure 5.5:** K-Nearest Neighbors example with 3 classes with uniform weights. [57]
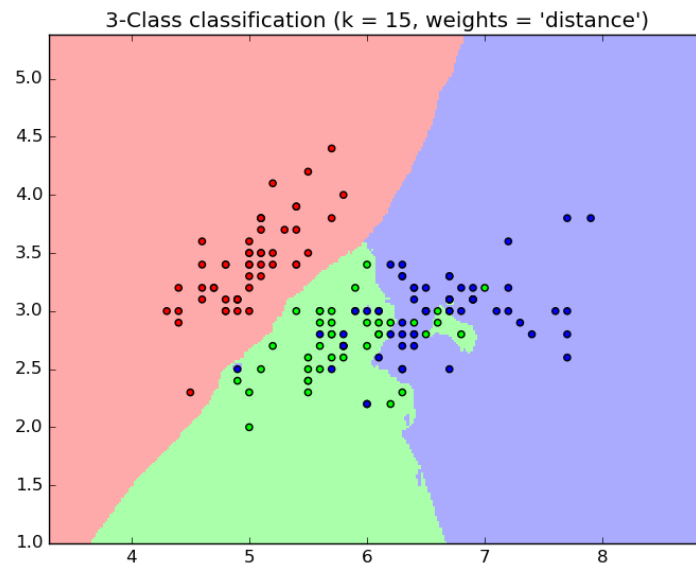


**Figure 5.6:** K-Nearest Neighbors example with 3 classes with inverse distance weights. [57]

### 5.2.2 Selection of parameter

The best choice of $k$ depends on the problem that is being solved and on the data that is being used to learn from. Larger values for $k$ reduce the effect of noise data during prediction. However, the other effect of large values of $k$ is that classes will have less distinct bounderies between them.

In general, the presence of noise severely limits the accuracy of the algorithm. Another issue that can limit the accuracy of the algorithm is irrelevant features. There are different methods to select and scale features in order to improve accuracy. One approach is to use evolutionary algorithms. [58]

## 5.3   Clustering

Clustering is a machine learning concept using unsupervised learning. Unsupervised learning does not have labels with the training set. An unsupervised machine learning algorithm tries to find structure within the given training set. Clustering is the first type of unsupervised learning. It tries to cluster the training samples into different clusters.

### 5.3.1   K-means Algorithm

The K-means algorithm is a simple clustering algorithm. K is the number of clusters that is going to be used. An example of how K-means could cluster data is shown in Figure 5.7. The algorithm first randomly places the K clusters in the space (from the training set). This can be done by randomly chosing K training samples. Then it repeats the following steps until the cluster centers remain stationary. it iterates over all training samples, and assigns them to the closest cluster. Next the cluster centers are moved to the center of the total cluster. [59]

```
# choose k random samples
c = choose_random_samples(k)
do:
    for sample in all_samples:
        c_i = closest_cluster_to(sample)
        c_i.append(sample)
    c.recalculate_center()
while cluster_center_changed(c)
```

The final clusters that are found by K-means are dependent on the random placement of the K clusters in the beginning. It could lead to suboptimal clustering. This can be fixed by running K-means a number of times and after each iteration check the value of the cost function to find the most efficient run of K-means. The cost function can be described as: [60]

$$J(c, \mu) = \frac{1}{m} \sum_{i=1}^{m} (||x^{(i)} - \mu^{(i)}||)^2 \tag{5.1}$$

There is a similar problem with determining the amount of clusters. The same method could be used to determine the correct number of clusters. However, manually determining the number of clusters could be more time efficient. [59]
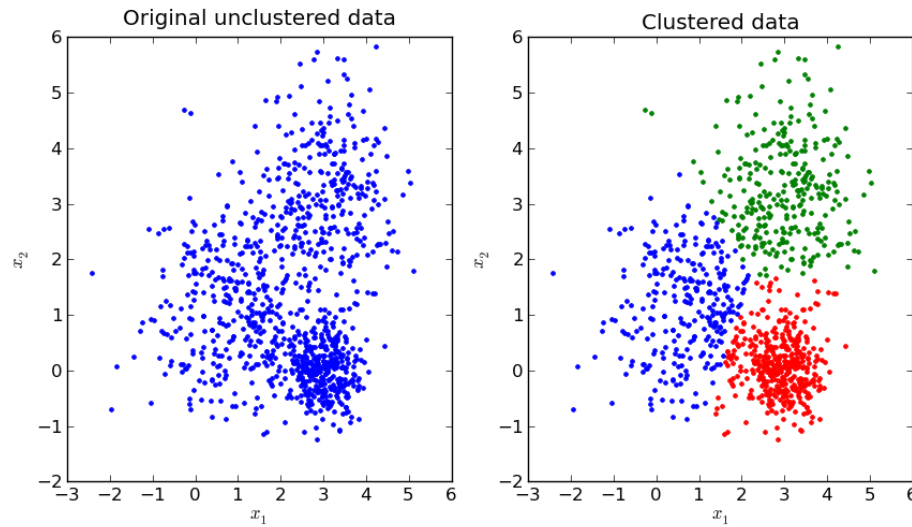
**Figure 5.7:** K-means clustering. [61]

## 5.4 One-class Support Vector Machine

Support Vector Machines can also be used for unsupervised learning. The SVM will only have one class. The SVM can be trained using a dataset filled with data that belongs to that one class. The SVM learns a decision boundary around these data points as seen in the example from Figure 5.8. This boundary can be used to predict whether a data point belongs to that class or whether it does not. One-class SVM's can also make use of kernels.
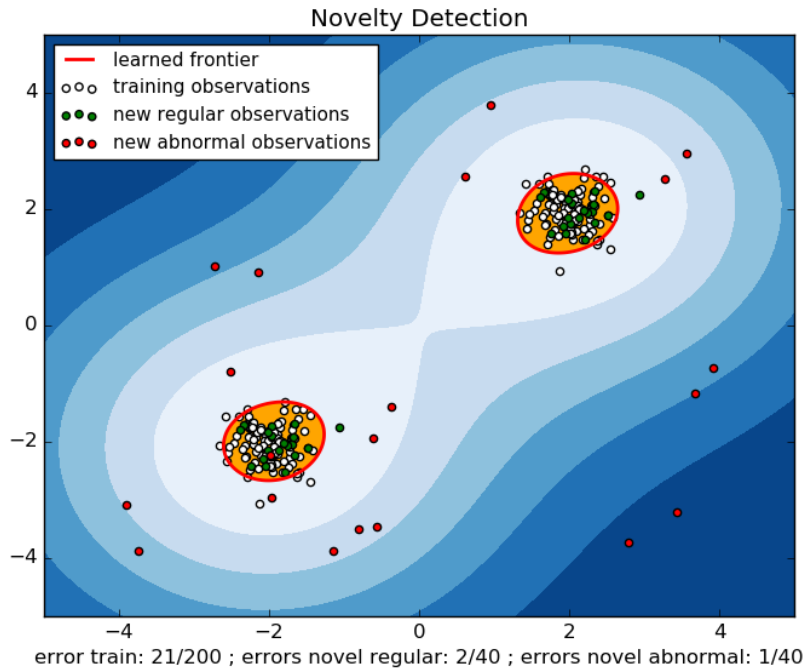
**Figure 5.8:** One-class SVM. [62]

## 5.5   Neural networks

Neural networks are a useful alternative to logistic regression if the amount of features becomes too large. The origin of neural networks are algorithms which try to mimic the brain. There is a hypothesis, the "one learning algorithm" hypothesis, that shows that the brain can learn very different things, such as sound, touch, etc. by using a single algorithm. [63]

A neural network is created of neurons, which are called a logistic unit. Each neuron receives input wires, and has an output wire, which computes a value using the sigmoid (logistic) hypothesis, the activation function. A neural network is a group of "neurons" that are connected together. This can be grouped into a layered approach as seen in Figure 5.9. The first layer is called the input layer, the final layer is called the output layer which outputs a $H_\theta(x)$ which is class. All layers inbetween these layers are called hidden layers. [64]
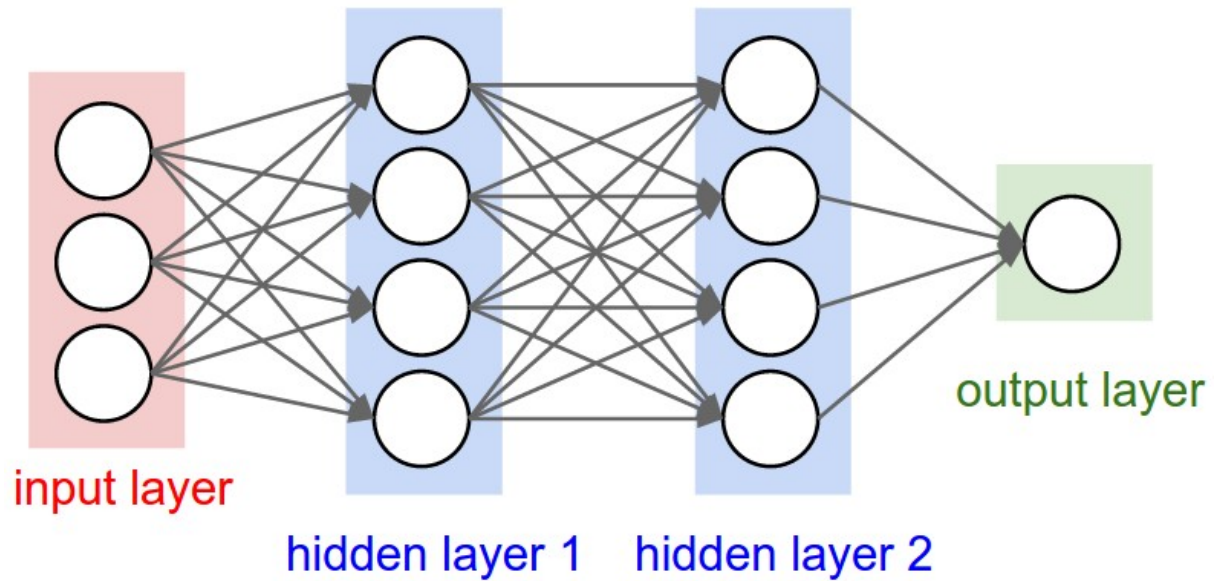
**Figure 5.9:** A neural network showing the different layers.[65]

Each logistic unit is denoted by $a_i^j$. $j$ is the layer and $i$ is the position in that layer. $\theta^j$ is a matrix of weights or parameters controlling function mapping from layer $j$ to layer $j+1$. $s_l$ is the number of units within a layer. The number of layers is denoted by $L$.

The neural network works similar to logistic regression, except that it performs logistic regression from layer to layer. The parameters $\theta_j$ required are learned by itself. The architecture of a neural network refers to the way the units are connected to eachother. Neural networks can be used for multi-class classification. Here, there are mutliple units in the output layer and each unit represents a different class. $K$ will denote the amount of output units. [66]
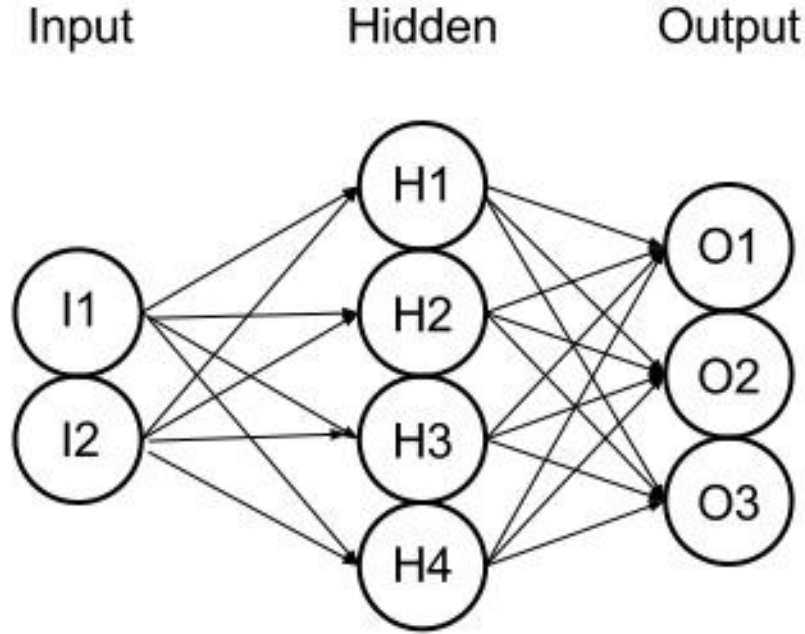
**Figure 5.10:** A neural network capable of multi-class classification. [67]

Data flows using the principle of forward propagation. The data passes through the first layers, moves to the second layers and so on, until it arrives at the final layer. Mathematically this can be described as:

$$a^{(1)} = x$$
$$a^{(2)} = g(\theta^{(1)} * a^{(1)})$$

$a^{(1)}$ is the input layer and $x$ are the features of a data point that are fed to the neural network. $g(\theta^{(1)} * a^{(1)})$ is the sigmoid logistic regression function with as input $\theta^{(1)} * a^{(1)}$. $\theta^{(1)} * a^{(1)}$ can be defined as $z^{(1)}$ to make the formulas simpler. This is a standard example of logistic regression. This happens for every node in a layer and for every layer. Every layer takes as input, the output from the previous layer. The data propagates forward through the network. [68]

### 5.5.1   Cost function and backpropagation

The cost function for neural networks is a generalisation of the cost function for logistic regression. The cost function accounts for the different layers, units and the number of output units. To minimize the cost function, the same methods such as gradient descent can be used. However, the problem is how to compute the partial derivative of the cost function. Using backpropagation, it is possible to compute this. [69]

The resulting class is known in the final layer. From here, the algorithm can find the error. $\delta_j^l$ will be the symbol used for the error of node $j$ in layer $l$.

The algorithm works backwards. It starts in the final layer. It calculates the error in this layer which is $\delta_j^{(4)} = a_j^{(4)} - y_j$ and starts working backwards. In the previous layer, the weighted sum is used of the

errors from the final layer. This is $(\theta^{(3)})^T \delta^{(4)}$. This happens from layer to layer and each time, the error is calculated using the $\theta$ values from that layer.

### 5.5.2 Using a neural network

A neural network should have as many input units as the dimension of features. The number of output units is equal to the number of classes. Default, there should be either 1 hidden layer or if there are more, all layers should have the same number of hidden units. [70] The neural network should be trained by first assigning random weights to the values of $\theta$. Afterwards, forward propagation should be used to get $H_\theta(x^{(i)})$. Next the cost function should be computed. Backwards propagation is used to compute the partial derivatives. The result of backwards propagation can be checked by numerical methods to compute the gradient. Finally gradient descent or other advanced optimization methods with back-propagation should be used to try to minimize the cost function as a function to the parameters $\theta$.

A neural network could be used to implement Autonomous Driving. The neural network can be trained using images which display a road. [71] The labeled data could be classes explaining whether the road goes straight, left or right. This could be used to make vehicles drive on a road, but could also be used to program drones to be able to navigate through forest paths. [72]

### 5.5.3 Deep neural networks

Deep Learning Algorithms are a modern modification to Artificial Neural Networks that exploit abundant cheap computation. Deep learning networks are very deep and complex neural networks. They contain a lot of hidden layers. Deep Boltzmann Machine (DBM), Deep Belief Networks (DBN) and Convolutional Neural Network (CNN) are examples of deap learning algorithms. These algorithms have specific rules aboout how the network is connected. [73]

## 5.6 Decision tree algorithms

Decision tree algorithms are a set of supervised learning algorithms used for classification and regression. The algorithms try to create a model that resembles a tree. Every prediction starts at the root of the tree. At every node, a certain decision rule is used. A decision rule is a condition, for example it could be: "Is feature A larger than 10". Once a leaf of the tree is reached the correct class or value is found. [74] An example of this can be seen in Figure 5.11
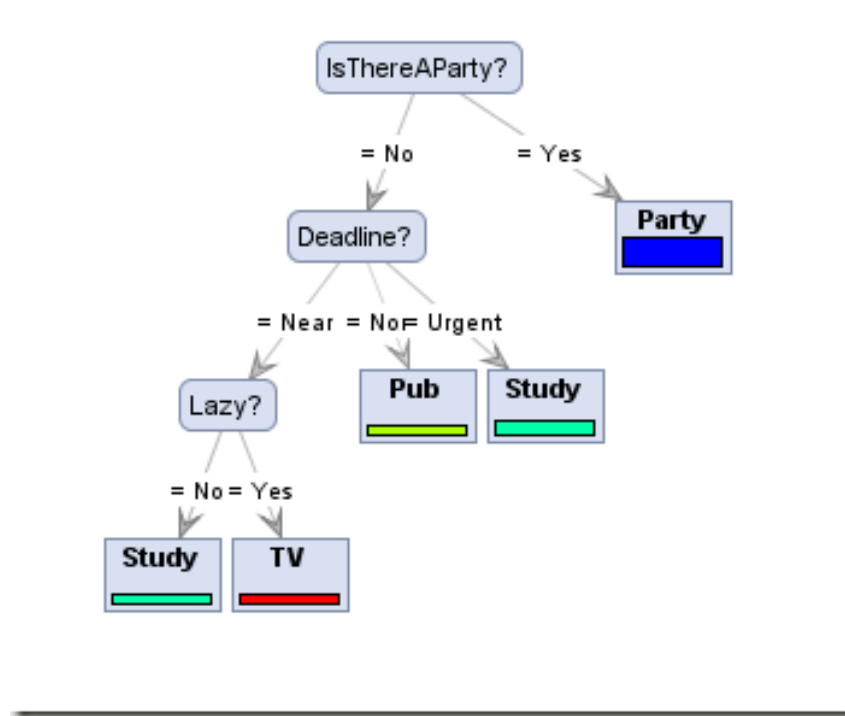
**Figure 5.11:** Decision Tree Algorithm example [75]

Descision tree algorithms usually work top-down. They chose a feature and look through the training data to check how many children need to be made. For example, in Figure 5.11, there is a feature "Deadline" which has 3 different values. In this case, the tree can create 3 children. In each child, the next feature is chosen and the process starts over until every training sample that would end up in this spot, all belong to the same class. When features are continuous, the algorithm tries to find ways to distinguish certain values. For example, for one feature, half of all training samples are larger than 100, the other half is smaller than 100. Then a check: "Is this feature larger or smaller than 100" might be chosen.

A decision tree is simple to understand and to visualise. They are also quite fast. However, a decision tree algortihm can easily create complex trees and overfit. They also cannot handle skewed data that well. [74]

## 5.7 Bayesian Algorithms

There are Bayesian Algorithms which explicitly apply Bayes' Theorem for problems such as classification.There are a lot of variances on Bayesian Algorithms, the most popular is Naive Bayes. [55] Naive Bayes is decent for classification but is not optimal for regression. [76]

Bayes Theorem is stated as follows [77]:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)} \tag{5.2}$$

$A$ and $B$ are events. $P(A)$ is the chance that event $A$ happens and $P(B)$ is the chance that event $B$ happens. $P(A|B)$, a conditional probability, is the probability of event $A$ given that $B$ is true. $P(B|A)$ is the probability of event $B$ given that $A$ is true. [77]

Naive Bayes is a supervised learning algorithm with the "naive" assumption that every feature is independent. The formula for the Bayes' Theorem is used for Naive Bayes. The event $A$ is a class. $B$ is the collection of features. [76]

In the Naive Bayes algorithm, for every class, the probability is calculated that that class belongs to the given sample for which a class is being predicted. The probability is: $P(class|x_0, x_1, ..., x_n)$. For this formula, the Bayes's Theorem can be applied. $P(x_0, x_1, ..., x_n)$ is constant since it only depends on the features. $P(x_0, x_1, ..., x_n|class)$ can be written as $P(x_0|class) * p(x_1|class) * ... * p(x_n|class)$ because of the naive assumption. All of these can be found during the training phase.

When training, $P(class)$ is the fraction of how much the class is represented in the training data. $P(x_i|class_j)$ for every feature can be found by looking at the training samples and looking at how much $x_i$ is represented when the class is $class_j$. When $x_i$ is not present in the training set, the distribution of the different values for $x_i$ are used to find out how much $x_i$ could be represented.

## 5.8 Association Rule Learning Algorithms

Association Rule Learning Algorithms are methods that extract rules that best explain observed relationships between variables in data. Apriori algorithm and Eclat algorithm are examples of such algorithms. These algorithms operate mainly on databases. [55]

In Figure 5.12, an example is given of a small database. Association Rule Learning could make several rules from this database. A rule could be "if a set contains alpha and beta, then there is a 50% chance that it also contains epsilon". [78] It does this by counting items. First it counts the frequency that alpha, beta, etc. appears in the database. Next, it counts the all the pairs $(alpha, beta)$. The algorithm continues until it cannot make any more sets. From these sets, it can make the association rules.



**Figure 5.12:** Apriori Algorithm example [78]

## 5.9    Specific algorithms for a sub-field

A lot of algorithms are specifically constructed for a specific sub-field of machine learning, for example computer vision, natural language processing, etc. Even within the categories of algorithms that were discussed, regression, regularization, instance-based, clustering and neural networks, there are a lot of different variants. However, these are considered to be to advanced and outside of the scope of this thesis.

# Chapter 6

# Machine Learning Techniques

## 6.1 Dimensionality reduction

Dimensionality reduction is the process of reducing the amount of features used in machine learning algorithms. This can be used to increase the accuracy and the performace of machine learning algorithms. One form is to do data compression. For example, transform 3D data into 2D data and eliminate a feature or dimension. It can also be used to reduce dimensions to be able to efficiently visualise data.

Dimensionality reduction can be used to speed up the time it takes for other learning algorithms to learn. By using dimensionality reduction, the amount of features or the amount of training samples is reduced which reduces the running time of the training, but the compressed data still retains the same information as the uncompressed data.

### 6.1.1 Principle Component Analysis

Principle Component Analysis is a way to do dimensionality reduction. The algorithm is formulated as a minimalisation problem. When given N-dimensional data and N-1 dimensional data is prefered, the algorithm tries to find the correct N-1 dimensional value so that the projection is the closest to the original data.

Before this algorithm is run, the features of the data should be scaled, so that all features are on a similar scale. This can be done by using mean normalization. In order to reduce the dimension from $n$ to $k$ the covariance matrix should be computed.

From this matrix, the eigenvectors need to be computed using singular value decomposition. From these values, only the first $k$ values are going to be used and be multiplied with the training data.

## 6.2 Anomaly detection

Anomaly detection is also a form of unsupervised learning. The algorithm learns what normal behaviour looks like through the training set and then tries to predict if a given input data belongs to the normal behaviour or is abnormal for any reason.

In Figure 6.1, the algorithm has been trained using the green data points. These make up the normal behaviour. The algorithm can then predict whether a data point is normal, or an anomaly. It seems very close to One-class SVM's as seen in Section 5.4. However, One-class SVM's are used for simple binary classification. Normal behaviour of a systems typically does not fall within a single class. Usually many different classes make up normal behaviour.
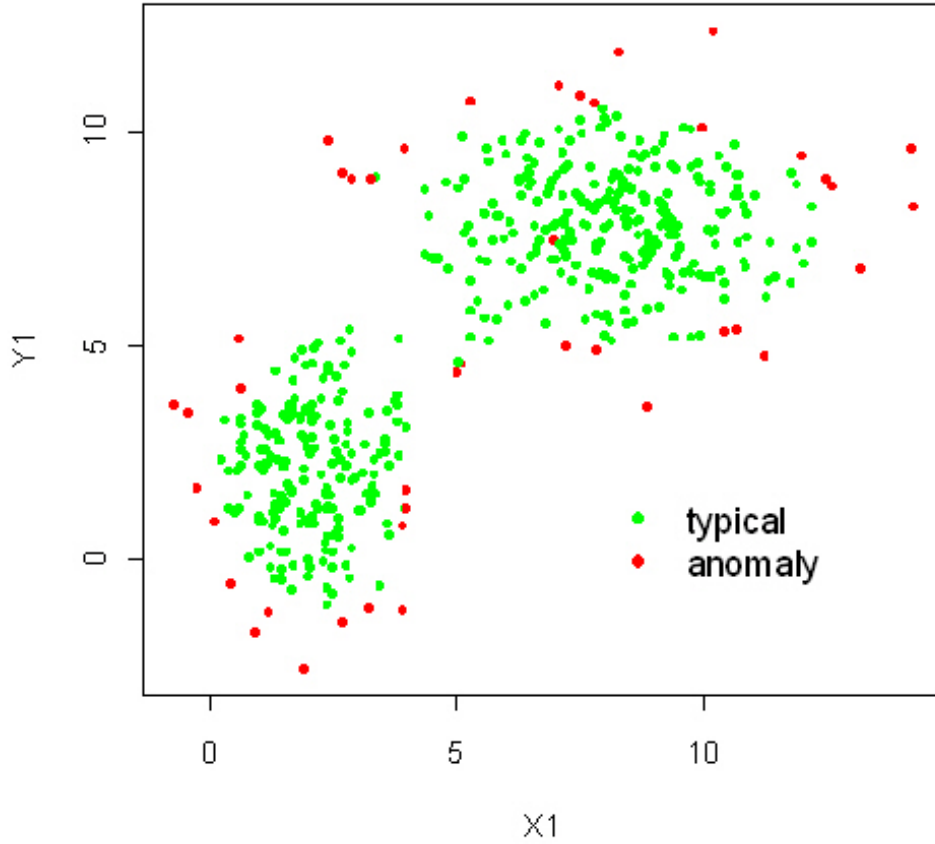
**Figure 6.1:** Anomaly detection. [79]

Anomaly detection is useful when there are a lot of data points belonging to normal behaviour and there are almost no abnormal behaviour data points. General supervised learning algorithms are usefull when there are a lot of data points of both normal and abnormal behaviour. [80]

### 6.2.1  Normal Distribution

Anomaly detection algorithms make heavy use of (Gaussian) Normal distribution:

$$x \sim N(\mu, \sigma^2) \tag{6.1}$$

Hereby $\mu$ is the mean parameter and $\sigma$ is the standard deviation. Now the probability of $x$ being an anomaly can be calculated as:

$$p(x) = \prod_{j=1}^{n} (p(x_j; \mu_j, \sigma_j)) \tag{6.2}$$

$$p(x_j; \mu_j, \sigma_j) = \frac{1}{\sqrt{2\pi}\sigma_j} exp(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}) \tag{6.3}$$

$p(x_j; \mu_j, \sigma_j)$ is the probability of $x_j$ with a Normal Distribution with parameters $(\mu_j, \sigma_j)$. $p(x)$ is the product of all these probabilities. The formula for the probability is the probability density of the Normal distribution. This function is plotted in Figure 6.2. The $y$-axis is the density function, the $x$-axis is the $x$ value in the formula.
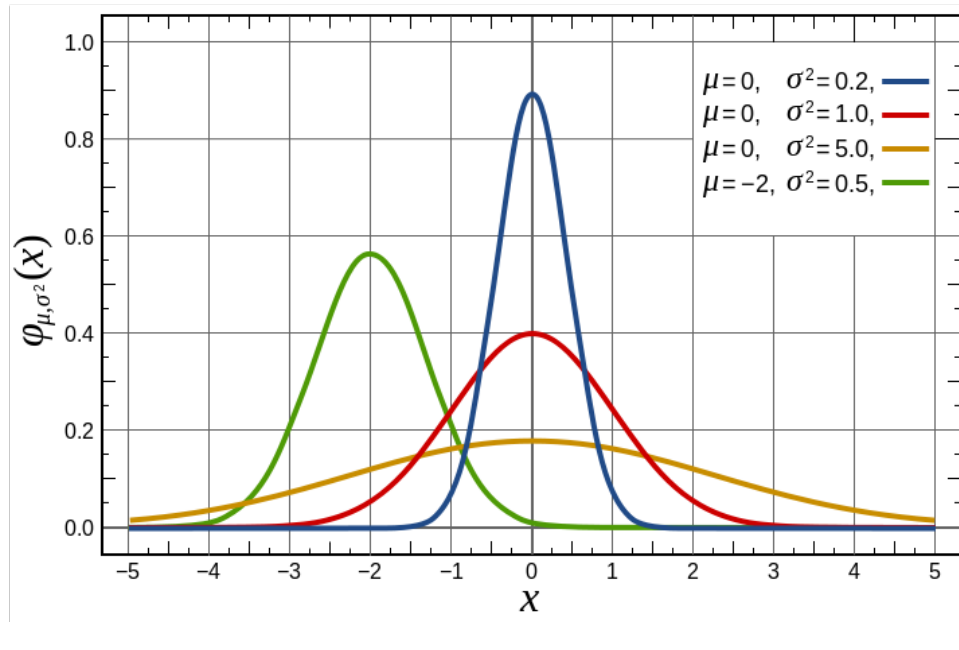
**Figure 6.2:** Normal Distribution. [81]

## 6.3 Online learning

Online learning is a technique to update an algorithm. Instead of feeding all data to the learning algorithm, the data is fed incrementally. The algorithm can keep learning from data even after it has initially learned a model.

Some algorithms such a KNN are immediatly able to do online learning. Other algorithms, such as SVM need to be slightly altered in order to be able to do online learning.

Online learning is very useful when the data is dependent on the time. For example, stock prices prediction. The data that is used to train an algorithm today might be able to predict stocks for tomorrow, but it is not very effective at prediction the stock market at long term. That is why it is useull to be able to keep training the algorithm. [82]

## 6.4 Bagging

Bagging is a technique which tries to create new datasets from a given data set. It takes a data set as input and generates multiple slight variations of the original data set. One method to do this is by generating a new dataset by sampling with replacement.

This means that to construct a new dataset of size $n$, $n$ elements are chosen from the original dataset and that doubles are allowed. These slight variations will have a different distribution of samples in the dataset. This has as effect that the learning algorithm is trained differently. This method is also called Bootstrapped Aggregation, or bagging for short.

Once multiple data sets have been generated, it is possible to combine multiple weaker models and training algorithms and try to combine the predictions made by these algorithms. This is useful to deal

with unstable data. Unstable data is data that can give different results depending on the algorithm that is used to process the data. An example of bagging and Neural Networks can be seen in Figure 6.3. [55]
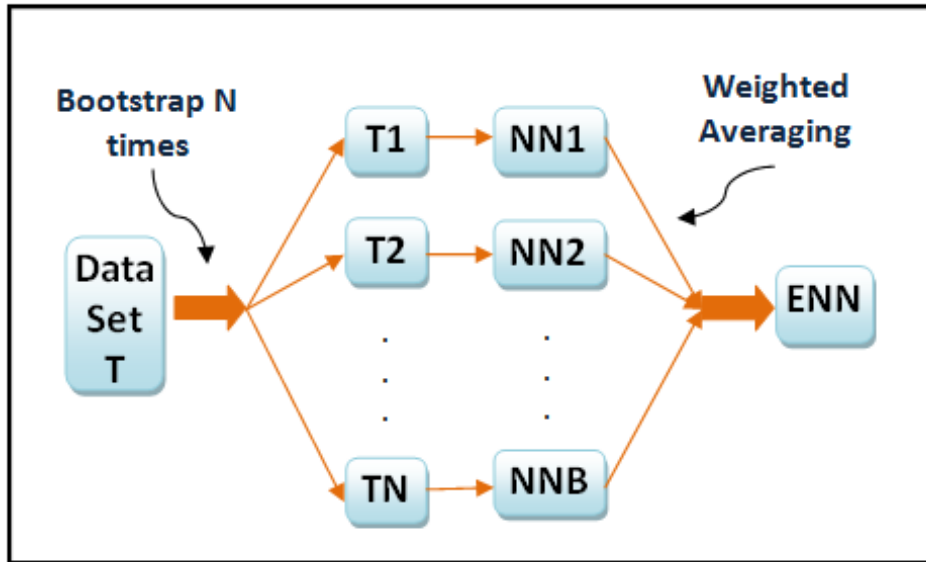


**Figure 6.3:** Bagging example with Neural Networks. [83]

# Chapter 7

# Validating an algorithm

## 7.1 Machine learning evaluation

Machine learning diagnostics are tests that can be run to get to know what is and isn't working with a machine learning algorithm. They also provide guidance as to how performance could be improved.

## 7.2 Error analysis

When trying to use machine learning for any purpose, such as intrusion detection systems, there are several considerations to be made. Another important step is the approach used to find the correct algorithm. The recommended approach is to start with a simple algorithm and test it with cross validation data. Afterwards, learning curves as seen in Section 7.2.4 could be plotted to decide if more or less data, more or less features, ... are likely to help. Finally, error analysis can be done by maually examining the samples on which the algorithm made mistakes. This could help to spot any systematic trends in the type of samples on which the algorithm is making mistakes.

The error analysis mostly consists of manual or brute force work. The samples on which the algorithm is wrong need to be categorized based on which features could help it categorize correctly and on which class it belongs to. Calculating statistics, such as the accuracy of the algorithm can also help with error analysis. It is possible to brute force this approach by automatically trying a lot of different combinations of features. This is less efficient than manually tweaking the algorithm.

Another issue to account for are skewed classes. Skewed classes are classes that are underrepresented in training data. For example, in binary classification, when trying to classify a flow as malicious or not, the training set might only provide $0.5\%$ malicious data. Knowing this, having an accuracy of $99\%$ does not seem that great.
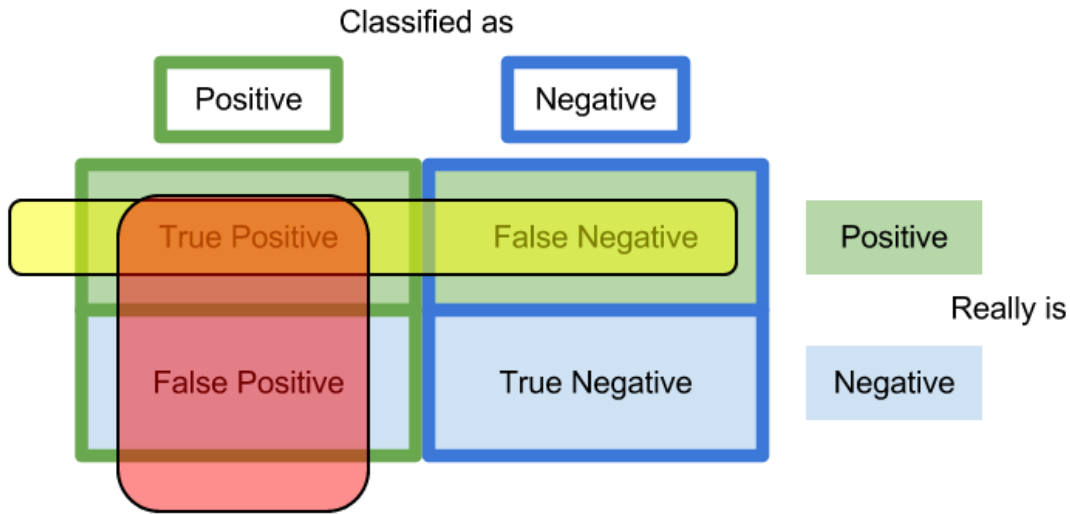
**Figure 7.1:** Precision and recall. [84]

A different metric is required to evaluate machine learning algorithms that are trained using skewed data. This metric is the precision and recall method. We classify a result as true positive, true negative, false positive and false negative. False positive and false negative respectively mean that the predicted value is falsely classified as positive and negative. These are the errors. True positive and true negative are correctly predicted values. **Precision** is defined as the fraction of correctly predicted positives. Precision is higher when there is a low amount of "false positives". Concretely high precision means that samples that are classified as positive, are actually positive:

$$\frac{truepositive}{truepositive + falsepositive}$$

**Recall** is defined as the fraction of predicted positives and the actual amount of positives. Concretely high recall means that samples that are actually positive are also classified as positive:

$$\frac{truepositive}{truepositive + falsenegative}$$

There is always a tradeoff to be made between precision and recall. As an effect of a higher precision, there will be a lower recall. Similarly, a higher recall means a lower precision.

Algorithms with a different precision and recall can be compared to eachother. This can be done using an F-score:

$$2\frac{PR}{P + R}$$

The F-score is the harmonic mean of precision and recall. For Table 7.1, this means that Algorithm 1 is the most effective. In contrast, using for example the average of both precision and recall would make Algorithm 3 the most effective.

The main reason to use the harmonic mean is because the average is taken of ratios (percentages), and in that case the harmonic mean is more appropriate than the (arithmetic) mean. [85]

**Table 7.1:** Example of precision and recall of certain algorithms.

|  | Precision (P) | Recall (R) |
|---|---|---|
| Algorithm 1 | 0.5 | 0.4 |
| Algorithm 2 | 0.7 | 0.1 |
| Algorithm 3 | 0.02 | 0.98 |

Looking at the values of the precision and the recall in Table 7.1, it can already intuitively be seen that Algorithm 1 is the best algorithm. For Algorithm 3, the precision is 0.02 and the recall is 0.98. This means that there were almost no false negatives. Every sample that is actually positive is also classified as positive. But the precision is very low, which means that there are a lot of samples that are actually negative, but were classified as positive. Concretely, Algorithm 3 classifies almost every sample as positive.

Algorithm 2 has similar problem but the other way around. Only Algorithm 1 seems to have some kind of balance between precision and recall which makes it seem as the best algorithm. In Table 7.2 the F-score for Algorithm 1 is the highest. This confirms that the F-score is a better metric than the average.

**Table 7.2:** Example of average and F-score of certain algorithms.

|  | Average | F-score |
|---|---|---|
| Algorithm 1 | 0.45 | 0.444 |
| Algorithm 2 | 0.4 | 0.175 |
| Algorithm 3 | 0.51 | 0.039 |

When evaluating machine learning algorithms for multi-class classification, the F-score is still used. The F-score is calculated for each seperate class. The average of these F-scores is then calculated. This gives a F-score for evaluating machine learning algorithms for multi-class classification. However, in the case of skewed data, the F-score can be deceiving when just the average is taken. This can be solved by using a weighted average. The weights are chosen based on how much a class is present in the data.

The data that is being fed into a machine learning algorithm is also important to consider. Sometimes a lot of data can be useful. First, the assumption is made that the features are chosen correctly and sufficiently. Lots of data is useful when a machine learning algorithm is used with a lot of parameters such as a neural network with a lot of hidden units. This means that the algorithm has low bias.

### 7.2.1 Different Algorithms

If the number of features is large relative to the number of training samples, then using a lineair kernel Support Vector Machine or logistic regression is preferred. A Gaussian kernel is preferred when there are more training samples than features, but the difference is rather small. Otherwise, new features should be added. Neural networks are almost always effective but they may be slower to train.

### 7.2.2 Evaluating the hypothesis

The most obvious way to test whether the hypothesis is correct is by dividing the training set into two sets. The first set which should have approximately 30% of the samples of the original training set will be used to train the learning algorithm. The other set is used to check whether the output of the learning algorithm is correct. This is done to make sure that there are enough testing samples. After the algorithm has done its learning with the training set, this check set is thrown into the algorithm. Aftwards the output of the algorithm is compared to the labels of the check set. If the output isn't correct, a test error

can be computed. These test errors can be used to compute global error value, which could be calculated by, for example, a mean squared error. This value can be used ito evaluate the hypothesis. [86]

### 7.2.3   Model selection algorithm

To get back to the problem of overfitting, there is another method next to regularisation called model selection. Model selection uses the same principle as mentioned above except it divides the training set into three new sets. A new training set, a cross validation set and a test set.

The training set is used to actually train the algorithm. The cross validation set is used to compare different algorithms or different models. Different models can be tested and compared to eachother by comparing the cross validation error, the error between the prediction and the actual output from the cross validation set. The testing set is used purely to test the algorithm once the cross validation has been done. It is considered good practice to use seperate sets for cross validation and testing. [87]

### 7.2.4   Diagnosing bias vs variance

High variance means that there is an overfitting problem. High bias on the other hand, is the opposite problem. It means that the hypothesis does not fit the training set at all. There is a relation between the amount of training samples and the training error. With more training data, the training error goes down.

With a low model complexity, there is a high error for the training set and a high error on the test set. This is a signal there is a underfitting problem or a high bias problem. It does not help to increase the amount of training samples since the model just is not complex enough.

With a high complexity model, the training set error is very low, but the test set error is very high. This is a signal there is an overfitting problem or a variance problem. This can be seen in Figure 7.2. The complexity of the model can be adjusted by changing the amount of features that are being used. [33]
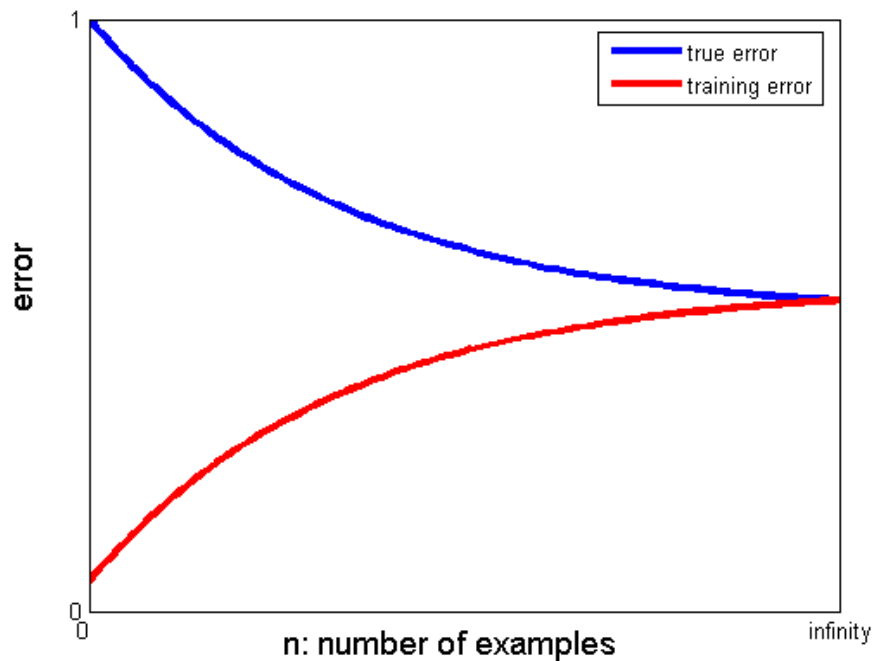
**Figure 7.2:** The effect of the amount of training samples on the accuracy with high bias. [88]

In Figure 7.2 the general trend of training error and test (or true) error is shown. This figure shows the example for high bias and no matter how many training samples there are, the functions are already converged to the same error amount.
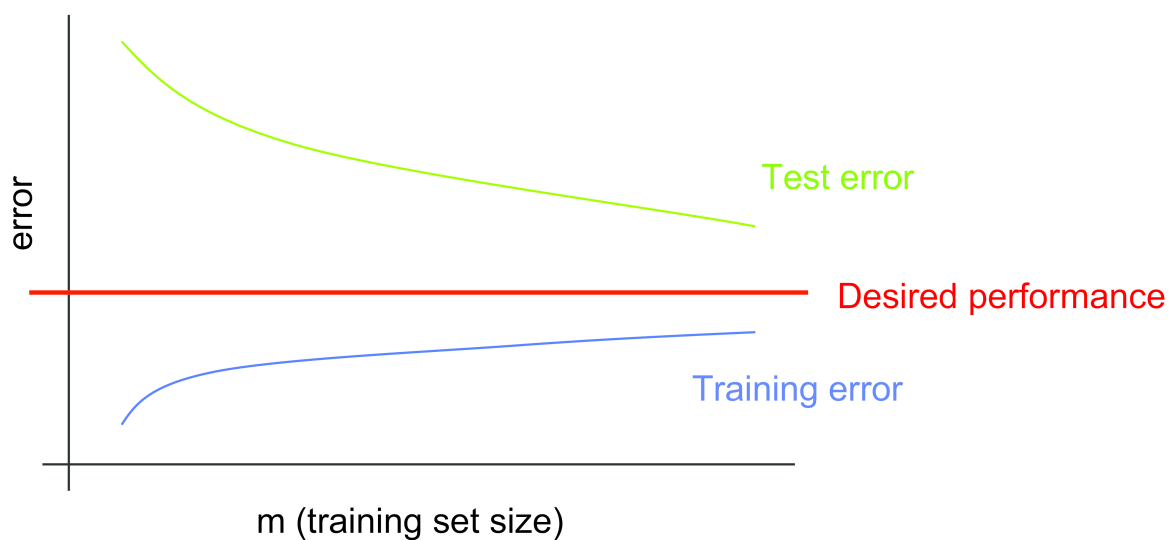


**Figure 7.3:** The effect of the amount of training samples on the accuracy with high variance. [89]

In contrary to high bias, high variance can be solved by using extra training samples. In that case there

is a gap inbetween the true error line and the training error line and it is likely that they still converge to the same point. This can be seen in Figure 7.3. With a bigger training set, they converge more and more until the "Desired performance" is reached. These curves are called learning curves.

# Chapter 8

# Machine learning for an IDS

An intrusion detection system has to detect whether some data it receives is either malicious or regular web traffic. This can be seen as a classification problem which means a machine learning algorithm for classification could be used. It needs to be determined whether data is either normal network traffic or, whether it corresponds to malicious behaviour.

Some parameters have to be chosen that will be feed into the machine learning algorithm. These parameters can be log files, entire packets, packet headers or IP flows as seen in Section 9. In general there are several advantages and disadvantages to using machine learning for intrusion detection.

## 8.1 Properties of using ML for an IDS

As said before, machine learning for an intrusion detection system is a classification problem. More precisely, it can be said that intrusion detection systems have to detect abnormal behaviour in a network with mostly normal behaviour. There are several problems that can be encountered when using machine learning techniques.

### 8.1.1 Network diversity

A first problem is the diversity of network traffic. The notion of "normal network traffic" is difficult to actually define. The bandwidth, duration of connections, origin of IP addresses, applications used can vary enormously through time. This makes it quite difficult for machine learning algorithms to distinguish between "normal network traffic" and malicious behaviour.[90]

However, this issue can be bypassed by defining what malicious behaviour looks like. If a machine learning algorithm learns what malicious behaviour looks like, it does not need to know anything about normal behaviour.

### 8.1.2 Detect new attacks

Another problem is the ability to detect new attacks. A machine learning algorithm compares incoming data with a model that it has created internally. An new type of malicious behaviour might appear to be closer to normal network traffic as compared to the model of known attacks. This is only an issue for completely new attacks since most attacks are similar to known malicious behaviour. The model that is created for malicious behaviour can better detect newer attacks than a signature based IDS. [90]

### 8.1.3 Classification

There is only one big remaining problem when trying to teach a machine learning algorithm what malicious behaviour looks like. Different types of malicious behaviour can look completely different. When the machine learning algorithm constructs a model, the model might not be able to generalise well or it might give a lot of false positives or false negatives.

There are several solutions that can be used in order to make machine learning algorithms more effective for intrusion detection systems. One option is to chance the way the classification problem is defined. Instead of defining the classes, "normal" and "malicious", there might be different classes for different types of malicious behaviour. In the same way, different classes can be defined for different types normal traffic.

This means that a different class is used for each different type of attack. This also makes it easy to define different severity levels for different attacks. Since the machine learning algorithm outputs the predicted class, a more detailed report can be generated on the anomaly. [90]

### 8.1.4 Learning

A big advantage of using machine learning is the learning part of the algorithm. The algorithm learns what malicious behaviour looks like. Whenever it has to predict whether something is malicious and what class it belongs to, it does not need to use perfect matching or heuristics such as other forms of intrusion detection as seen in Section 2.4. It has learned a model that describes what the different classes of malicious behaviour look like. [91]

## 8.2 Evaluating ML for an IDS

With a machine learning algorithm, performance can be measured using the F-score as seen in Section 7.2. However, for intrusion detection systems, this is not enough by itself. The F-score assumes that recall and precision has the same importance. This is not necessarily the case when evaluating intrusion detection systems. [91]

In Table 8.1, it is shown when there is a true positive and true negative. A false positive occurs when a sample is actually Normal but is classified as an Intrusion. A false negative occurs when a sample is actually an Intrusion but is classified as Normal.

**Table 8.1:** Performance measure.

|  | Intrusion | Normal |
| --- | --- | --- |
| **Intrusion** | True Positive | False Negative |
| **Normal** | False Positive | True Negative |

A false negative is bad, since it means that an Intrusion was not detected. But most IDS's are used in a layered approach. This means that if one layer does not detect an Intrusion, another layer might detect it.

The top layer usually looks at the entire network traffic. Because the entire network traffic might be huge, the data that is processed is kept to a minimum. This means that the top layer will use IP flows. This means that not detecting an Intrusion is not that horrible. A low recall could be disastrous if the amount of detected Intrusions is huge, which can happen if the amount of data that is passed through the IDS is also huge.

The layered approach might also work completely different. Perhaps the first layer tries to detect as many anomalies as possible (and having a low recall) and then passing the data for which anomalies have been detected to other layers. This approach means that a low recall is not bad.

The scoring used for an IDS that uses machine learning is dependant on how the IDS is going to be used. [91]

## 8.3 Using ML for an IDS

Data has to be processed before it can be used within a machine learning algorithm. This means that features have to be chosen. Some features can be easy to find, other have to found by experimenting and running tests.

Using all the features of a dataset does not necessarily guarantee the best performances from the IDS. It might increase the computational cost as well as the error rate of the system. This is because some features are redundant or are not usefull for making a distinction between different classes. [92]

# Chapter 9

# IP Flows

Flows are aggregated from all packet data that travels through the network. Flow exporters are programs which collect network packets and aggregate them into flow records. A flow is not the same as a TCP connection. A flow can be any communication between two devices with any protocol. Flows are defined using a (source_IP, destination_IP, protocol) tuple. This is why flows are also called IP Flows.

Since flow data does not contain any payload information, intrusion detection systems that use flow data cannot detect malicious behaviour embedded within payload data. [13] A flow is constructed using a flow exporter. These constructed flows are usually collected and stored within a flow collector. An example of this can be seen in Figure 9.1.
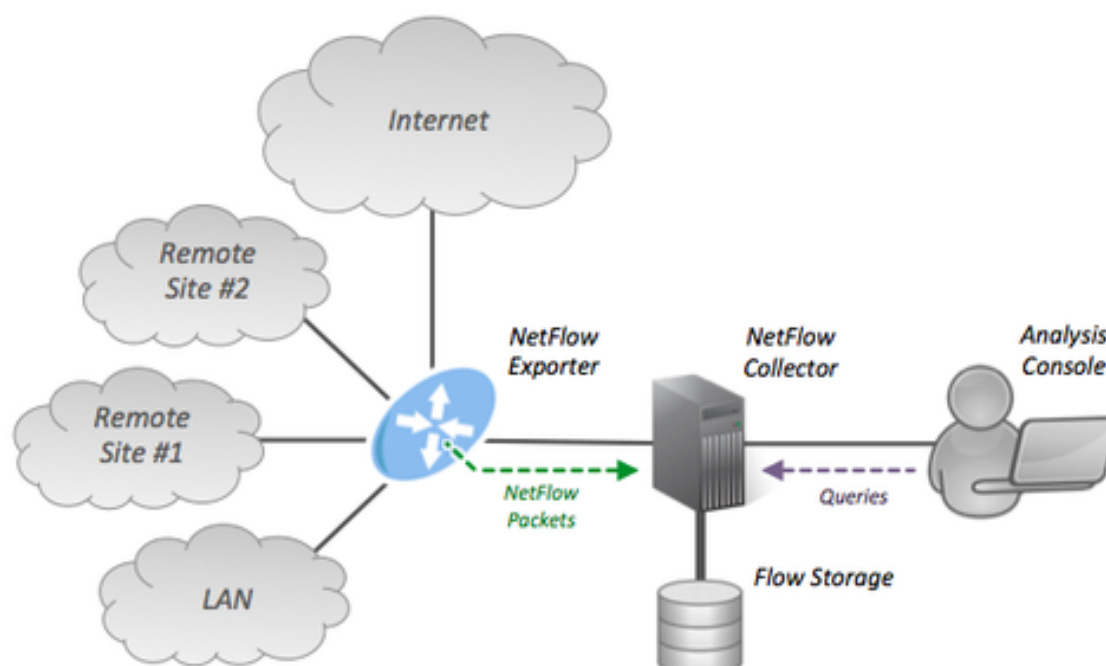


**Figure 9.1:** Netflow. [93]

## 9.1 Attributes of a flow

The following attributes are available within a flow:

- Source IP

- Destination IP

- Protocol name

- Source port

- Destination port

- Starting time of the flow

- Duration of the flow

- Amount of packets in the flow

- Amount of bytes in the flow

These attributes are most commonly available within a flow. However there are different standards for a flow. Some contain less information, some contain more information. Should a flow exporter be implemented, some additional features can be generated from packet data.

- Amount of TCP SYN within the flow

- Source and Destination Type of Service

- TCP flags set

These extra attributes can be very usefull when trying to determine what is normal and abnormal behaviour.

The **source IP** is the IP address of the device or network that started the flow. The **destination IP** is the IP address of the device or network that the sender connected to. The IP address can be either an IPv4 or an IPv6 address. If the flows were gathered and created on a host computer, the IP address field can be a MAC address instead of an actual IP address. These type of flows are generated by newer flow exporters.

The **source port** is the port from which the flow originates. The **destination port** is the port to which the flow connects to.

The **protocol name** is the name of the protocol that is used for the flow. Each flow uses only one protocol. Some flow exporters might provide the protocol is numerical form, others in ASCII format. The numerical form is used mostly by flow exporters that only allow explicit IP traffic. Other flow exporters also contain non-IP traffic such as ICMP messages.

Flow attributes such as the **amount of packets** and the **amount of bytes** are self-explanatory. They are usefull to know for example whether the flow contains many packets which each contain almost no payload, or whether the flow contains one gigantic packet.

The **starting time** is the time when the flow is started. This is saved in unix time. The **duration of the flow** is the time between the start and end of a connection. When the flow is a TCP connection, the start and end times are defined. These are the moments that a SYN is sent and a FIN ACK is received. However, should the flow be an UDP flow for example, there is no clear end. The flow terminates itself after a timeout.

## 9.2   Using IP flows

In order to be able to use IP flows, data has to be extracted from the flows. In general, the different attributes from a flow are already useful. However it could also be interesting to extract other information from the flows.

### 9.2.1 Country of origin

The country of origin is something that can be extracted from flow information. The source and the destination IP can be given to an IP API which gives back information about the physical location of the IP address. Also the region, city name and ISP can be extracted. An example of such a query can be seen in Figure 9.2. This API is just one example of the available API's on the web. [94]

```json
{
  "status": "success",
  "country": "United States",
  "countryCode": "US",
  "region": "CA",
  "regionName": "California",
  "city": "San Francisco",
  "zip": "94105",
  "lat": "37.7898",
  "lon": "-122.3942",
  "timezone": "America\/Los_Angeles",
  "isp": "Wikimedia Foundation",
  "org": "Wikimedia Foundation",
  "as": "AS14907 Wikimedia US network",
  "query": "208.80.152.201"
}
```

**Figure 9.2:** IP API JSON format. [94]

### 9.2.2 Time

The starting time can also be manipulated. From the unix starting time, information such as the year, month, day, weekday, hours and so on can be found. This information makes it easier to categorize network traffic and allows certain patterns to be seen. For example, it might be that certain types of web traffic only happen at a certain weekday.

# Chapter 10

# Implementation

## 10.1 Technology stack

The main library that was used is *scikit-learn* [95]. *Scikit-learn* is a robust machine learning library for Python. It is build upon NumPy, SciPy, and matplotlib. It is also open source and commercially usable with the BSD license. There is a lot of documentation available for *scikit-learn*. This library was chosen since the library offers the most important algorithms, the documentation. The programming language, Python, is also good for prototyping.

Most algorithms that have been explained in Section 5 are implemented in *scikit-learn*. The stable version of this library does not implement neural networks. However, the unstable version does include neural networks.

*Scikit-learn* also contains different methods to visualise machine learning algorithms such as a graph to show the learning curve. These can be a useful tool to evaluate the performance of machine learning algorithms. It also contains methods to calculate the F-score. This is useful since that means that mistakes when calculating the F-score are less likely to happen.

### 10.1.1 Program execution

The implementation works in different steps. A JSON config file is used to define the elements that are used within the program. This contains the data to be used for learning, for checking, the machine learning algorithm, etc. A full explanation of the config file as well as a general user guide can be found in Appendix A.

Once the config file has been read, the program can start the training phase. In this phase the specified algorithm is used and trained using the given data. Afterwards the prediction phase starts. This phase uses the prediction data and gathers all results. The structure of the program and the modules reflect these different phases.

### 10.1.2 Structure

The implementation is build to be modular. New components can easily be added. Details on how to add new components to these modules is explained in the developer guide in Appendix B. The first module is the machine learning module. This module contains all machine learning algorithms that can be used.

There is also a feature module. This module contains the available classes that can be used to extract features from the flows. A loader module contains all classes required to load the data from the different datasets.

A training module contains the different classes used for training. These classes use a loader class and

pass the data to the machine learning algorithm. They define which data is supposed to be used (for example, using only abnormal behaviour and leaving out the normal behaviour).

Finally there is a results module. This module receives all the output from the machine learning algorithms and has to log these or visualise them.

## 10.2 Datasets

In order to test the implementation and the algorithms, different datasets were used. Each dataset is used to test a different aspect of the machine learning algorithms. First, a subset of a dataset has to be chosen to be fed to the machine learning algorithms for learning. Afterwards, using the method seen in Section 7.2.2, the algorithm is tested using another subset of the same dataset.

In the third step, the algorithms are tested using real-world data that is labeled. Finally, in the fourth step, the algorithms are tested using raw, unlabeled real-world data. This is to make sure that the algorithm performs well on unprocessed real-world data. Several datasets have been used to test the machine learning algorithms.

### 10.2.1 CTU-13 Dataset

The CTU-13 dataset has been used for steps one to three for the testing of the machine learning algorithms. This is a labeled dataset. It contains botnet behaviour, normal and background traffic. The data was captured in the CTU University, Czech Republic, in 2011. It consists of thirteen different captures, each of which run a different botnet malware. Figure 10.1 shows the amount of data within each capture. Note that the captured data is only from a couple hours. The flows within the dataset contain extra information. Each flow also contains the TCP flags obtained by OR-ing the TCP flags field of all packets of the flow. [96]

| Id | Duration(hrs) | # Packets | #NetFlows | Size | Bot | #Bots |
|----|---------------|-----------|-----------|------|-----|-------|
| 1 | 6.15 | 71,971,482 | 2,824,637 | 52GB | Neris | 1 |
| 2 | 4.21 | 71,851,300 | 1,808,123 | 60GB | Neris | 1 |
| 3 | 66.85 | 167,730,395 | 4,710,639 | 121GB | Rbot | 1 |
| 4 | 4.21 | 62,089,135 | 1,121,077 | 53GB | Rbot | 1 |
| 5 | 11.63 | 4,481,167 | 129,833 | 37.6GB | Virut | 1 |
| 6 | 2.18 | 38,764,357 | 558,920 | 30GB | Menti | 1 |
| 7 | 0.38 | 7,467,139 | 114,078 | 5.8GB | Sogou | 1 |
| 8 | 19.5 | 155,207,799 | 2,954,231 | 123GB | Murlo | 1 |
| 9 | 5.18 | 115,415,321 | 2,753,885 | 94GB | Neris | 10 |
| 10 | 4.75 | 90,389,782 | 1,309,792 | 73GB | Rbot | 10 |
| 11 | 0.26 | 6,337,202 | 107,252 | 5.2GB | Rbot | 3 |
| 12 | 1.21 | 13,212,268 | 325,472 | 8.3GB | NSIS.ay | 3 |
| 13 | 16.36 | 50,888,256 | 1,925,150 | 34GB | Virut | 1 |

**Figure 10.1:** Amount of data and botnet type for each capture. [96]

Each capture contains only a small amount of botnet samples as seen in Figure 10.2. Most flows are background flows. This is expected of botnet behaviour since it does not generate an large amount of network traffic. The actual labeling is much more detailed as compared to Figure 10.2. Each flow is labeled with its exact source. This could be google analytics, google webmail or a windows update. The flows within the dataset only contain the regular information that is found within netflow. The abnormal behaviour within this dataset is internal abnormal behaviour. In the evaluation chapter, this dataset is called the CTU dataset.

| Scen. | Total Flows | Botnet Flows | Normal Flows | C&C Flows | Background Flows |
|---|---|---|---|---|---|
| 1 | 2,824,636 | 39,933(1.41%) | 30,387(1.07%) | 1,026(0.03%) | 2,753,290(97.47%) |
| 2 | 1,808,122 | 18,839(1.04%) | 9,120(0.5%) | 2,102(0.11%) | 1,778,061(98.33%) |
| 3 | 4,710,638 | 26,759(0.56%) | 116,887(2.48%) | 63(0.001%) | 4,566,929(96.94%) |
| 4 | 1,121,076 | 1,719(0.15%) | 25,268(2.25%) | 49(0.004%) | 1,094,040(97.58%) |
| 5 | 129,832 | 695(0.53%) | 4,679(3.6%) | 206(1.15%) | 124,252(95.7%) |
| 6 | 558,919 | 4,431(0.79%) | 7,494(1.34%) | 199(0.03%) | 546,795(97.83%) |
| 7 | 114,077 | 37(0.03%) | 1,677(1.47%) | 26(0.02%) | 112,337(98.47%) |
| 8 | 2,954,230 | 5,052(0.17%) | 72,822(2.46%) | 1,074(2.4%) | 2,875,282(97.32%) |
| 9 | 2,753,884 | 179,880(6.5%) | 43,340(1.57%) | 5,099(0.18%) | 2,525,565(91.7%) |
| 10 | 1,309,791 | 106,315(8.11%) | 15,847(1.2%) | 37(0.002%) | 1,187,592(90.67%) |
| 11 | 107,251 | 8,161(7.6%) | 2,718(2.53%) | 3(0.002%) | 96,369(89.85%) |
| 12 | 325,471 | 2,143(0.65%) | 7,628(2.34%) | 25(0.007%) | 315,675(96.99%) |
| 13 | 1,925,149 | 38,791(2.01%) | 31,939(1.65%) | 1,202(0.06%) | 1,853,217(96.26%) |

**Figure 10.2:** Distribution of labels in CTU 13 Dataset. [96]

It is also important to note that each capture contains different botnet behaviour. In Figure 10.3, the characteristics of the different botnet captures are shown. During experiments, it is important that the machine learning algorithm is trained and tested using data with the same characteristics. This is because each characteristic exhibits unique behaviour. The labeling of the flows is also different between the different captures. This makes it difficult to train with one capture and test with another capture. Even among the captures with the same botnet characteristics, there is different labeling between the captures. For this reason, it was chosen that the algorithm would be trained and tested using a single capture. The capture itself is chosen at random. This is done to make sure that experiments were done across different captures. Using every capture would explode the training set and make testing unpractical.

| Id | IRC | SPAM | CF | PS | DDoS | FF | P2P | US | HTTP | Note |
|----|-----|------|----|----|------|----|-----|----|------|------|
| **Table 2 — Characteristics of the botnet scenarios. (CF: ClickFraud, PS: Port Scan, FF: FastFlux, US: Compiled and controlled by us.)** | | | | | | | | | | |
| 1 | √ | √ | √ | | | | | | | |
| 2 | √ | √ | √ | | | | | | | |
| 3 | √ | | | √ | | | | √ | | |
| 4 | √ | | | | √ | | | √ | | UDP and ICMP DDoS. |
| 5 | | √ | | √ | | | | | √ | Scan web proxies. |
| 6 | | | | √ | | | | | | Proprietary C&C. RDP. |
| 7 | | | | | | | | | √ | Chinese hosts. |
| 8 | | | | √ | | | | | | Proprietary C&C. Net-BIOS, STUN. |
| 9 | √ | √ | √ | √ | | | | | | |
| 10 | √ | | | | √ | | | √ | | UDP DDoS. |
| 11 | √ | | | | √ | | | √ | | ICMP DDoS. |
| 12 | | | | | | | √ | | | Synchronization. |
| 13 | | √ | | √ | | | | | √ | Captcha. Web mail. |

**Figure 10.3:** Characteristics of botnet captures. [96]

## 10.2.2  Tracelabel Dataset

This dataset has been used for steps one to three for the testing of the machine learning algorithms. The tracelabel dataset is a database constructed at the University of Twente in September 2008. It contains flow data of traffic collected from a honeypot that was positioned in the network of the university. The honeypot ran sevaral network services such as ftp, ssh, http, etc. The honeypot only captured abnormal behaviour. As such, all of the data within the dataset is actual abnormal behaviour. [97]

The flows within the dataset contain extra information. Each flow also contains the TCP flags obtained by OR-ing the TCP flags field of all packets of the flow. In Table 10.1, the labeling that is used within this dataset can be seen. The abnormal behaviour within this dataset is external abnormal behaviour. In the evaluation chapter, this dataset is called the SQL dataset.

**Table 10.1:** Labeling of the Tracelabel dataset.

| Id | label | Explanation |
|----|-------|-------------|
| 1 | ssh_scan | An ssh scan |
| 2 | ssh_conn | Unauthorized ssh connection attempt |
| 3 | ftp_scan | An ftp scan |
| 4 | ftp_conn | Unauthorized ftp connection attempt |
| 5 | http_scan | An http scan |
| 6 | http_conn | Unauthorized http connection attempt |
| 7 | authident_sideeffect | Unauthorized identification attempt |
| 8 | irc_sideeffect | Suspicious irc traffic |
| 9 | icmp_sideeffect | Suspicious icmp traffic |

## 10.2.3  EDM Dataset

This is a dataset of unlabeled netflow data gathered from the EDM network. The EDM or the Expertise centre for Digital Media, which is located at the science park at UHasselt. The EDM is the ICT research institute of Hasselt University and has around 47 researchers working there. The dataset is gathered from the entire EDM network. The flows are gathered at the edge of the network and contain both incoming and outgoing flows.

This dataset has been used for step four for the testing of the machine learning algorithms, to check

if the machine learning algorithms perform well on unprocessed real-world data. It spans from the 18th of Februari, 2016 till the 24th of March 2016. The data spans from 10am till midnight. This dataset is used mainly to check whether the machine learning algorithms perform well on real-world data.

### 10.2.4 Cegeka Dataset

A dataset from Cegeka has been used. Cegeka is a full-service ICT-business with offices in Belgium, the Netherlands, Poland and Romania. Cegeka has three datacenters which are all located in Belgium. They are located in Hasselt, Leuven and Veenendaal. The dataset contains network data from the datacenter in Hasselt.

The dataset spans three days, from the third of April till the fifth of April. This dataset contains un-labeled netflow data. It also contains logs from firewalls. These logs can be used to check whether the machine learning algorithm has a good performance.

## 10.3 Feature selection

A lot of information can be found within a flow as explained in Section 9. The standard attributes of a flow have been chosen to be used as features. The extra information such as the TCP flags that are set have not been used in every step of the evaluation process since they are not available in all of the datasets that have been used. However, they have been used in some experiments to test their effects using the datasets that do have this information.

In the implementation, the features are all seen as continuous data. In order to use discrete data, a conversion has to be made to allow them to be used as continuous data. Both source and destination IP addresses are used as features. They were used as contiguous data by using the IP address in decimal format. Different features have been used for IP addresses and MAC addresses.

Another option was to look at IP addresses as if they were discrete features. For example, it could be possible to only look at the three least significant bytes of the IP address. This is a simple method to divide IP addresses into "subnets". However, this is not exactly a correct subnet. The best method would be to use the longest prefix matching to find the subnet to which an IP address belongs. However, it is difficult to find this information. Another method would be to use something like the country of origin. Using an API to find out the country of origin of an IP could be used to make a discrete feature. This divides IP addresses in different regions similar to a subnet which makes it a good subsitute for longest prefix matching.

The source and destination ports are also used as features. Ports are not continuous information. The difference between using port 21 and port 22 is just as big as the difference between using port 22 and 1024. This means that ports are discrete data. One method to solve this is to make a different feature for each port. The possible values for each feature would be 1 or 0. However, this has as effect that there are a lot of features, in the tens of thousands. This is very inefficient and impractical. The solution that has been chosen in to make a feature that determines whether the port is one of the 1024 well known ports or another port.

These protocols are all IP protocols. Protocols which are encapsulated within an IP header. They mainly belong to the transport layer, but can also contain network layer protocols such as ICMP. Protocols are also discrete data. However, there are not that many different protocols. This means that it is viable to use a different boolean feature for each protocol.

The amount of packets and the amount of bytes in the flow are both continuous by nature. They can just be used as continuous features. The duration is also continuous and is represented as such.

The ability to remove this limitation is behind a paywall. The starting time of a flow has not been used. Manipulations of the starting time as described in Section 9.2.2 has not been used since there is not enough data for it to be useful. There is only training data from a timeframe of a couple hours. Different data sets were used for the training and the starting time of the flows do not match between these data sets. This means that, since the starting time is not consistent, results from this is also not consistent or representative. However, such a feature does seem promising if the training data is consistent enough.

To conclude, there are three sets of features that are used. The first set of features, to which will be reffered to as the standard feature set is ($src\_port$, $dst\_port$, $duration$, $src\_ipv4$, $src\_ipv6$, $src\_mac$, $dst\_ipv4$, $dst\_ipv6$, $dst\_mac$, $total\_packets$, $total\_bytes$). The standard feature set is used as the "baseline", it is used to compare against the other feature sets. The second feature set is ($src\_port$, $dst\_port$, $duration$, $src\_ipv4$, $src\_ipv6$, $src\_mac$, $dst\_ipv4$, $dst\_ipv6$, $dst\_mac$, $total\_packets$, $total\_bytes$, $TCP\_flags$). This feature set contains the exact same information as the standard feature set but also contains the TCP flags and is called the TCP feature set. The final feature set is ($src\_port$, $dst\_port$, $duration$, $src\_country$, $dst\_country$, $total\_packets$, $total\_bytes$). This set replaces the IP addresses by the country information and will be called the country feature set.

## 10.4 Algorithm selection

Both supervised and unsupervised algorithms have been used. The algorithms from Section 5 are the most common algorithms. Before more complex algorithms such as deep neural networks should be used, the more common and general algorithms should be tested.

### 10.4.1 Unsupervised learning

K-Means clustering is used in order to test whether results can be found using clustering algorithms. K-means is a simple clustering algorithm and already gives an indication whether a problem can be solved using clustering, or whether clustering offers no advantage. However, no method was found to verify whether to clusters that the K-means algorithm made were correct.

One-class Support Vector machines are used in an attempt to use binary classification. They are quite fast in execution. They were used to find out whether it is a viable technique to preprocess incoming data and check whether a One-class Support Vector machine finds it to be abnormal behaviour before passing it to other algorithms.

### 10.4.2 Supervised learning

Support vector machines have been used in the implementation. It is an popular algorithm and can do both linear and non-linear classification which make it a promising choice to test in the implementation.

K-nearest Neighbors was the most promising algorithm. This algorithm is used extensively through throughout the implementation and the tests. The fact that the classification happens on basis of the different neighbors instead of trying to make a classifier seemed to fit the feature data better.

Neural networks also seemed very promising. However, the stable version of the library *scikit-learn* does not implement neural networks. The unstable version does include neural networks.

Through the study of different machine learning algorithms, decision tree algorithms and Bayesian algorithms have also been discussed. They seemed less promising for the problem of intrusion detection. The difference between a normal flow and an abnormal flow is very slight and it seemed that these algorithms would make more mistakes. They are still used in the implementation to find out whether this assumption is correct or not.

## 10.5 Comparision to MIT AI²

In Section 2.5.3, a paper from MIT was discussed. Their system has several similarities with the system proposed in this thesis. Their system uses supervised learning algorithms and could detect around $85\%$ of all attacks. This is quite difficult to compare to results from this thesis. They focused on using log files in order to detect attacks. Not much is known about which kind of attacks they can detect. Even though both systems use a different source for the intrusion detection, the detection engine has several similarities.

As said before both systems use supervised learning algorithms as the main form of intrusion detection. The difference is that their system uses the supervised learning as a form of feedback learning.

Log files are processed and fed to a unsupervised learning algorithm, which already makes some predictions. These predictions are manually verified and then fed to the supervised learning algorithm for the final predictions. The system in this thesis does not use such a feedback system.

This thesis tries to find an effective method of intrusion detection that can happen completely automatically, without manual interference. This means that this thesis disregarded such methods of feedback. However, the idea that samples could be "preprocessed" using unsupervised learning algorithms is something that has been looked at. Unfortunately, no good methods were found to use the unsupervised learning algorithm as can be seen in Section 11.7

During the experiments done for this thesis, there was the problem of finding labeled data. The MIT researchers prefered to use unsupervised machine learning in a preprocessing step since labeled data is rare and attacks constantly evolve. This reasoning could be considered the big weakness of the system proposed in this thesis. Most algorithms that have been tested, require a big amount of training samples in order to be trained effectively.

## 10.6 Source code and experiment results

The source code for the framework that was developed for this thesis has been open-sourced and is hosted on Github. The repository can be found at [98]. The results from the experiments can also be found on this repository. The repository does not contain the datasets that were used.

# Chapter 11

# Evaluation

In this chapter, the different machine learning algorithms are evaluated. In Section 10.2, the different evaluation steps are explained. There are four steps. Step one is the training of the machine learning algorithm, this is evaluated using learning curves. Step two is evaluation within the same dataset, this is done using the CTU dataset. In the third step, cross-dataset validation is done. The algorithms are evaluated by using multiple datasets containing real-world data. In the final step, unlabeled real-world data is used.

For the first and second step, the algorithms are always trained using a subset from the datasets and tested using another subset. Since the datasets contain a lot of different labels (and thus classes), it has been chosen not to always follow the "30/70" rule as explained in Section 7.2.2. Instead the algorithm is trained using the number of training samples that indicates a good accuracy in the learning curves, and tested using a set of 230000 samples. This number was found by looking at all the learning curves from the different algorithms and choose the appropriate training size. From these training sizes, the largest value was chosen and on this value the "30/70" rule was used. This was done to make sure that every experiment was tested using the same amount of samples and that for each experiment the balance between training and testing samples was at least "30/70". This is done to allow statistically correct comparisons to be made across different machine learning algorithms. Each training set also contains an about even split between malicious and non-malicious samples. The samples themselves are chosen at random from the datasets.

In the third step, the algorithm is trained using samples from both the CTU dataset and the SQL dataset. The amount of training and testing samples are calculated on the same way as they were calculated for the first and second step. The SQL dataset has not been used in the second step. This dataset only contains malicious samples, most of these samples are also ssh scans. This means that when the machine learning algorithm is only trained using data from this dataset, it only knows what an ssh scan looks like, and predicts that everything is a ssh scan since it does not know any other classes.

In the fourth step, the algorithm is evaluated using real-world data. This is done using the Cegeka and EDM dataset. The Cegeka dataset also contains firewall logs, which have been used to check the performance of the algorithm. The EDM dataset has also been used and the samples have manually been checked. This was a tedious and slow task and the reason that not many samples have been used from this dataset. In total 10000 samples were used from the EDM dataset.

The verification that was done only verified whether the flow was malicious or non-malicious. The exact class that was predicted has not been verified. This means that a flow could be labeled as a google-analytics flow but in reality could be some other non-malicious flow. This manual check was done by using the labeled data sets and comparing the flows manually. The SSH scans within the SQL dataset all look very similar. It can be seen whether the flow looks similar or not. For example, an indicator that was used was the amount of packets and the amount of bytes in a flow. Other indicators were the protocol that was used and the TCP flags that were set. For the botnet traffic, an easy indicator was the whether the flow was outgoing or incoming. The botnet traffic in the dataset was often outgoing. If it wasn't it was always characterised by having a low amount of packets ($<= 2$) and a low amount of

bytes ($< 500$).

As explained in Section 10.3, there are several ways to choose features. All these different ways have been used in the experiments. In order to remove the factor that chance might play a role in the experiments, each experiment has been **run 10 times**. In total, more than a thousand experiments were done. The results have been averaged, and the variance is calculated to assess statistical relevance.

In order to evaluate the machine learning algorithms, some metrics need to be used. The F-score as explained in Section 7.2 is used. The F-score is always a number between $0$ and $1$, indicating a percentage. The F-score can be used in several ways. Either it can be used to evaluate how well a machine learning algorithm can distinguish between malicious and non-malicious flows, or it can be used to evaluate how well the algorithm can classify flows.

In order to use the F-score to evaluate how well a machine learning algorithm can distinguish between malicious and non-malicious flows, the intrusion detection needs to happen binary. All that needs to be known is whether a flow is malicious or it is not. In order to calculate the F-score, it needs to be known what a true positive and true negative is. A true positive is defined as a malicious flow, a true negative is defined as a non-malicious flow. This type of F-score will be called the **binary F-score** in this thesis.

The F-score can also be used to check how accurate the machine learning algorithm can predict the correct class. The interest here is in multi-class classification. This F-score will be called the **multi-class F-score**. A flow is positive when it belongs to the correct class, and negative when it does not belong to the correct class.

In the implementation, multi-class classification is used. The machine learning algorithm predicts the actual class of a flow. The implementation also has a list which contains all classes that are considered non-malicious. Using this list, it can be calculated whether the class that was predicted means that the flow is malicious or non-malicious.

A high binary F-score does not mean that the multi-class F-score is also high. The machine learning algorithm might be able to distinguish between malicious flows and non-malicious flows, but might not see the difference between different anomalies.

**A baseline** needs to be established. Comparing each machine learning algorithm, a best algorithm might be found. However, if the algorithm does not perform better than the baseline, the algorithm does not perform well. Two baselines have been used in this thesis. The first baseline predicts the classification randomly. This is used to know whether a machine learning algorithm can predict the actual class better than randomly. The other baseline always predicts a non-malicious class. This baseline is used since the non-malicious samples outnumber the malicious samples. This means that having a couple false negatives is worse than a couple false positives.

Another metric that has been used are **learning curves** as seen in Figure 7.2.4. These cannot be used to measure the performance as they only show the accuracy of the algorithm. The **accuracy** is the percentage of correctly classified flows. The reason that it cannot be used to measure the performance is because the accuracy is not an accurate representation of the performance. For example, the accuracy might be 99%. However it might be possible that every sample was classified as one of the non-malicious classes. This means that even though the accuracy is 99%, the algorithm cannot detect any malicious behaviour.

In order to construct the learning curve, a subset is chosen from the datasets. The subset is chosen such that the distribution of the labels is kept. The size of the subset depends on which algorithm is being evaluated and how long it takes for that algorithm to train. In each iteration in the construction of the learning curve, random samples are chosen to create the training set during that iteration. This

has as effect that not necessarily every label appears in the training set. For this reason, it was chosen to use the accuracy in the learning curve and not the F-score. Since these F-scores are not necessarily a good representation of the F-scores that are found in the other experiments. Even though the training set might not contain every label, the constructed learning curve can still show signs of overfitting or underfitting.

Figure 11.1 shows an example of a learning curve. The lines show how the accuracy behaves when the amount of training samples are increased. The $y$-axis shows the accuracy as a number between $0$ and $1$ which indicate a percentage. The lightly colored areas are calculated using a standard deviation. The red line represents the training score. This is how accuratly the algorithm can make predictions on the data that was used to train the algorithm. The green line, is data that is used for cross-validation. This is data that was not used for training. The learning curves of the baselines have been included in all learning curve graphs.

It can be seen that in the beginning the random baseline has a higher accuracy than when more training samples are used. It would be expected that the random baseline has the same accuracy across different sizes of training samples. However, the random baseline can only predict labels that appear in the training set. Since some labels are skewed, they may not appear in the training set. Because of this, the random baseline will predict only the non-skewed labels and have a slightly higher accuracy. When the training set is larger, the chance that not every label appears in the training set is very small. The effect of this is also very small, with just a difference of $0.8\%$. The training sets used for the normal experiments are casefully chosen so that every label appears in high quantities in order to properly train the algorithms.

For the learning curves, this does not matter, since the learning curve will look the same and still show the signs of overfitting and underfitting. This has been verified by manually constructing a learning curve. The library used for the implementation uses the random approach when constructing learning curves and also immediatly shows the standard deviation. The library can also construct these learning curves faster and uses a lot of iterations.
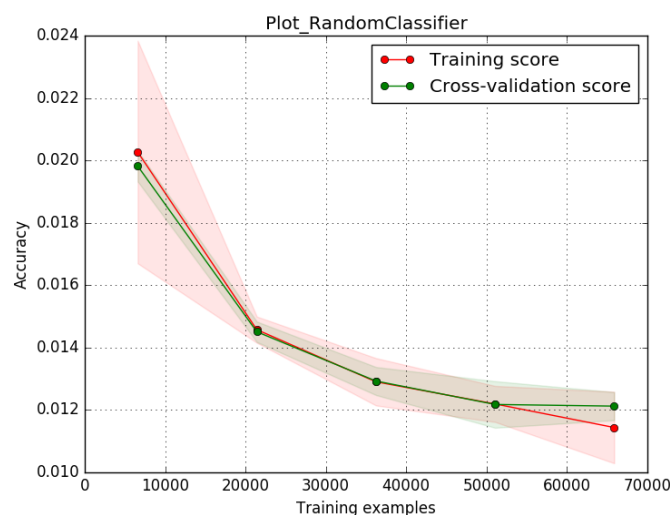


**Figure 11.1:** Random Baseline learning curve

In Table 11.1 and Table 11.2 the scores are listed. The binary precision of Table 11.2 would give a divison by zero. If that is the case, the score is set to $0.0$. Both F-scores in both baselines are low. The random algorithm is terrible at classification. The closer to $0$ the F-score is, the less well the algorithm performs.

The tables also list the amount of samples used, aswell as the binary classification. This is done to show how many samples were used in the test. The amount of positive and negative training samples is also shown. Finally, the variance is shown. This shows how much the F-scores fluctuate in the different experiments that were done.

| **Table 11.1:** Baseline Random classifier (uniform): Average. | |
| --- | --- |
| Multi-class F-score | 0.0162 |
| Multi-class Precision | 0.1205 |
| Multi-class Recall | 0.0096 |
| Binary F-score | 0.4653 |
| Binary Precision | 0.3592 |
| Binary Recall | 0.6603 |
| Total amount of samples | 231797.0 |
| False negative | 28278.0 |
| False positive | 98054.15 |
| True negative | 50492.85 |
| True positive | 54972.0 |
| Positive training samples | 82159.0 |
| Negative training samples | 49638.0 |
| Variance Multi-class F-score | 1.18e-07 |
| Variance Multi-class Precision | 8.33e-06 |
| Variance Multi-class Recall | 4.46e-08 |
| Variance Binary F-score | 8.21e-07 |
| Variance Binary Precision | 5.00e-07 |
| Variance Binary Recall | 2.48e-06 |

| **Table 11.2:** Baseline Random classifier (All negative): Average. | |
| --- | --- |
| Multi-class F-score | 0.0287 |
| Multi-class Precision | 0.1104 |
| Multi-class Recall | 0.0183 |
| Binary F-score | 0.0 |
| Binary Precision | 0.0 |
| Binary Recall | 0.0 |
| Total amount of samples | 231797.0 |
| False negative | 83250.0 |
| False positive | 0.0 |
| True negative | 148547.0 |
| True positive | 0.0 |
| Positive training samples | 0.0 |
| Negative training samples | 49638.0 |
| Variance Multi-class F-score | 3.41e-07 |
| Variance Multi-class Precision | 4.51e-06 |
| Variance Multi-class Recall | 1.14e-07 |
| Variance Binary F-score | 0.0 |
| Variance Binary Precision | 0.0 |
| Variance Binary Recall | 0.0 |

## 11.1 K-nearest Neighbors

K-nearest Neighbors as seen in Section 5.2 is a very promising algorithm for intrusion detection. $k$ was chosen at the default value of $5$. However, different experiments have been done to compare the effect of $k$ on the perofrmance of the machine learning algorithm. The default distance metric that was used, is the Manhattan distance. In a realistic data-set, the malicious data is skewed. For this reason, the solution proposed in Section 5.2.1 was used. The first experiment that was done, was to construct the learning curve as seen in Figure 11.2.

Since KNN uses the $k$-closest neighbors and their distance to the to-be-predicted data point, it is normal that the training score is $1.0$. This is because the training score is calculated by feeding the training data to the machine learning algorithm to test it. The closest point to any training data point is that data point itself. This means that the algorithm will predict the same class. Overfitting is unlikely. In that case, the training score is expected to be high, but the cross-validation score should go down. It can be seen that the algorithm converges to an accuracy of about 86%, but the algorithm shows a general trend to improve the accuracy when more training data is used. Underfitting is also unlikely because of this trend.
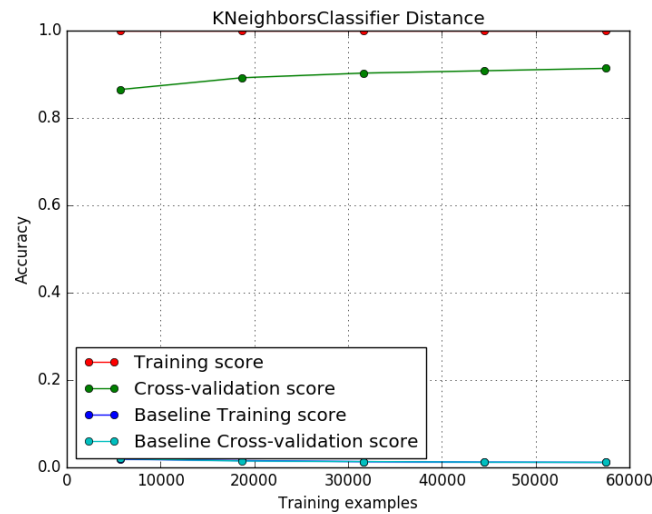
**Figure 11.2:** Learning Curve for K-Nearest Neighbors

The first experiment using the F-score that has been done was a validation within the CTU dataset. This is shown in Table 11.3. The experiment was done with all three feature sets. The algorithm was trained using $\sim 80000$ samples with about as much malicious and non-malicious data. First the general results are discussed. Afterwards the different feature sets are discussed.

The variance on both F-scores are very low, which means that the average result is statistically relevant. The binary F-score is $\sim 0.9500$ is quite high. This means that the algorithm is able to distinguish between malicious and non-malicious data. The multi-class F-score is lower, which means the algorithm does make more mistakes concerning the exact classification. Together this means that the malicious data is grouped together, but the actual classes within the malicious data are less distinguishable.

The effect of using the country of origin does not seem to have a lot of impact on the results. This could be due to the fact that the data belongs to a data set which contains data from mosty the same country. Using TCP flags does increase the performance immensely. This means that the different classes do become more distinguisable when more data, and more specifically the TCP flags, are used. KNN uses the $k$-closest neighbors and their distance to the to-be-predicted data point. Because of the high binary and multi-class F-score, it can be said that most classes and more specifically, the malicious and non-malicious samples are grouped together. If different samples weren't grouped together, then the $k$-closest neighbors would be of different classes, and the F-score would be much lower.

**Table 11.3:** K-Nearest Neighbors: comparing different feature sets: CTU-dataset

| Feature set | Standard | TCP | Country |
|---|---|---|---|
| Multi-class F-score | 0.6153 | 0.8476 | 0.6149 |
| Multi-class Precision | 0.6820 | 0.8527 | 0.6863 |
| Multi-class Recall | 0.6280 | 0.8482 | 0.6278 |
| Binary F-score | 0.9500 | 0.9857 | 0.9499 |
| Binary Precision | 0.9204 | 0.9797 | 0.9202 |
| Binary Recall | 0.9816 | 0.9918 | 0.9816 |
| Total amount of samples | 195519.0 | 195519.0 | 195519.0 |
| False negative | 1048.0 | 466.5 | 1049.0 |
| False positive | 4830.0 | 1169.2 | 4841.0 |
| True negative | 133798.0 | 137458.8 | 133787.0 |
| True positive | 55843.0 | 56424.5 | 55842.0 |
| Positive training samples | 35363.0 | 35363.0 | 35363.0 |
| Negative training samples | 49638.0 | 49638.0 | 49638.0 |
| Variance Multi-class F-score | 1.23e-32 | 1.23e-32 | 1.23e-32 |
| Variance Multi-class Precision | 0.0 | 1.23e-32 | 1.23e-32 |
| Variance Multi-class Recall | 1.23e-32 | 0.0 | 1.23e-32 |
| Variance Binary F-score | 0.0 | 1.23e-32 | 1.23e-32 |
| Variance Binary Precision | 0.0 | 0.0 | 4.93e-32 |
| Variance Binary Recall | 4.93e-32 | 0.0 | 1.23e-32 |

Table 11.4 shows cross-dataset validation. It is interesting to see that the F-score is consistently higher than the F-scores from Table 11.3 . The positive samples consist of 30000 samples from the SQL dataset and 35000 samples from the CTU dataset. The amount of samples used for prediction consists of 50000 samples from the SQL dataset and $\sim$ 190000 samples from the CTU dataset.

The multi-class F-score is still high, which means that the algorithm is still able to correctly classify samples, even though data from different datasets are mixed. The high binary F-score means that the algorithm is able to correctly distinguish between malicious samples and non-malicious samples. The effect of the different feature sets is still the same in this experiment.

**Table 11.4:** K-Nearest Neighbors: comparing different feature sets: Cross-dataset

| Feature set | Standard | TCP | Country |
|---|---|---|---|
| Multi-class F-score | 0.7571 | 0.8798 | 0.7616 |
| Multi-class Precision | 0.7768 | 0.8839 | 0.7784 |
| Multi-class Recall | 0.7628 | 0.8803 | 0.7693 |
| Binary F-score | 0.9631 | 0.9924 | 0.9713 |
| Binary Precision | 0.9543 | 0.9892 | 0.9544 |
| Binary Recall | 0.9720 | 0.9956 | 0.9889 |
| Total amount of samples | 246467.0 | 246467.0 | 246467.0 |
| False negative | 3015.1 | 474.5 | 1200.6 |
| False positive | 5013.9 | 1172.2 | 5083.8 |
| True negative | 133614.1 | 137455.8 | 133544.2 |
| True positive | 104823.9 | 107364.5 | 106638.4 |
| Positive training samples | 65363.0 | 65363.0 | 65363.0 |
| Negative training samples | 49638.0 | 49638.0 | 49638.0 |
| Variance Multi-class F-score | 2.23e-08 | 1.20e-09 | 7.29e-07 |
| Variance Multi-class Precision | 4.99e-09 | 1.39e-09 | 7.62e-07 |
| Variance Multi-class Recall | 4.43e-08 | 5.77e-10 | 8.90e-07 |
| Variance Binary F-score | 3.76e-08 | 5.20e-12 | 7.79e-09 |
| Variance Binary Precision | 1.35e-08 | 2.35e-15 | 1.33e-08 |
| Variance Binary Recall | 1.65e-07 | 2.06e-11 | 2.63e-08 |

During previous experiments, the standard feature set was used to find out how KNN performs and how the different feature sets weight in on the performance. In Table 11.5, experiments were done using different values for $k$, the chosen values were $3$, $5$ and $7$. The results show that both $k = 3$ and $k = 7$ perform better than $k = 5$.

In order to be absolutely certain that no mistakes happened, the entire experiment (all 10 runs) was done again twice. The results remained the same. From this, it can be concluded that $k = 5$ is a local minimum for the performance in relation to $k$. It can also be concluded that the actual value of $k$ needs to be determined experimentally when such a system would be set up in a data center since both $k = 3$ and $k = 7$ perform the same.

**Table 11.5:** K-Nearest Neighbors: comparing different values of K: Cross-dataset

| K | 3 | 5 | 7 |
|---|---|---|---|
| Multi-class F-score | 0.8783 | 0.7571 | 0.8807 |
| Multi-class Precision | 0.8831 | 0.7768 | 0.8846 |
| Multi-class Recall | 0.8778 | 0.7628 | 0.8819 |
| Binary F-score | 0.9924 | 0.9631 | 0.9926 |
| Binary Precision | 0.9892 | 0.9543 | 0.9897 |
| Binary Recall | 0.9956 | 0.9720 | 0.9956 |
| Total amount of samples | 246467.0 | 246467.0 | 246467.0 |
| False negative | 474.5 | 3015.1 | 474.5 |
| False positive | 1172.2 | 5013.9 | 1117.4 |
| True negative | 137455.8 | 133614.1 | 137510.6 |
| True positive | 107364.5 | 104823.9 | 107364.5 |
| Positive training samples | 65363.0 | 65363.0 | 65363.0 |
| Negative training samples | 49638.0 | 49638.0 | 49638.0 |
| Variance Multi-class F-score | 1.18e-09 | 2.23e-08 | 3.96e-10 |
| Variance Multi-class Precision | 8.69e-10 | 4.99e-09 | 5.43e-10 |
| Variance Multi-class Recall | 5.48e-10 | 4.43e-08 | 1.90e-10 |
| Variance Binary F-score | 3.23e-11 | 3.76e-08 | 5.19e-12 |
| Variance Binary Precision | 1.47e-14 | 1.35e-08 | 1.30e-10 |
| Variance Binary Recall | 1.28e-10 | 1.65e-07 | 3.09e-11 |

Different experiments were also done to compare the effect of different distance metrics on the performance. The metrics that were used were the Manhattan distance, the Euclidean distance, the Chebyshev distance and the Canberra distance. These distances were introduced in Section 4.5. The results of these experiments are shown in Table 11.6.

The Manhattan distance performed the least. The Euclidean distance and the Chebyshev distance performed the same. The Canberra distance which is a weighted Manhattan distance has the best performance when looking at multi-class classification. The fact that the Canberra distance performs good and the Manhattan distance does not can be due to the fact that the Canberra distance is weighted. This means that the different features are not on scale which is indeed the case.

The Canberra distance, however, is not the best when looking at the binary classification. Both the Euclidean distance and the Chebyshev distance perform better. Chebyshev works using the $max$ distanc between the features from two samples. In the Canberra metric, each feature has the same "importance". This could be the reason that Chebyshev works better for binary classification. Not every feature has the same importance. Chebyshev only uses the $max$ distance which most likely belongs to a feature which is more important for binary classification.

It can be concluded that depending on the features that are chosen, different distance metrics need to be used. This means that when new features are added, new experiments should be done to find out which distance metric works the best.

**Table 11.6:** K-Nearest Neighbors: comparing different distance metrics: Cross-dataset

| Distance metric | Manhattan | Euclidean | Chebyshev | Canberra |
|---|---|---|---|---|
| Multi-class F-score | 0.7571 | 0.8772 | 0.8754 | 0.9151 |
| Multi-class Precision | 0.7768 | 0.8814 | 0.8796 | 0.9216 |
| Multi-class Recall | 0.7628 | 0.8778 | 0.8761 | 0.9122 |
| Binary F-score | 0.9631 | 0.9917 | 0.9912 | 0.9768 |
| Binary Precision | 0.9543 | 0.9882 | 0.9874 | 0.9594 |
| Binary Recall | 0.9720 | 0.9951 | 0.9950 | 0.9950 |
| Total amount of samples | 246467.0 | 246467.0 | 246467.0 | 539.2 |
| False negative | 3015.1 | 528.4 | 539.2 | 4540.7 |
| False positive | 5013.9 | 1281.4 | 1369.2 | 4540.7 |
| True negative | 133614.1 | 137346.6 | 137258.8 | 134087.3 |
| True positive | 104823.9 | 107310.6 | 107299.9 | 107299.8 |
| Positive training samples | 65363.0 | 65363.0 | 65363.0 | 65363.0 |
| Negative training samples | 49638.0 | 49638.0 | 49638.0 | 49638.0 |
| Variance Multi-class F-score | 2.23e-08 | 3.63e-10 | 1.98e-09 | 1.56e-09 |
| Variance Multi-class Precision | 4.99e-09 | 6.78e-10 | 2.73e-09 | 1.25e-09 |
| Variance Multi-class Recall | 4.43e-08 | 2.09e-10 | 1.41e-09 | 1.49e-09 |
| Variance Binary F-score | 3.76e-08 | 1.43e-10 | 3.95e-11 | 1.94e-09 |
| Variance Binary Precision | 1.35e-08 | 7.70e-14 | 6.93e-11 | 6.49e-10 |
| Variance Binary Recall | 1.65e-07 | 5.68e-10 | 9.37e-11 | 5.38e-09 |

As a final step, KNN was evaluated using a real-life unlabeled dataset. This was done using the Cegeka and EDM dataset. In Table 11.7, the results can be seen. The algorithm was tested on $\sim$ 11000000 samples. From these samples 10000 were from the EDM dataset. From these samples, there were only 3 malicious flows. The algorithm predicted 24 malicious botnet flows, all of which were false positive. This could be seen because the class that was predicted belonged to a class that was always outgoing traffic. These samples however, were incoming traffic. The algorithm predicted the SSH scans correctly, but also predicted 19 other SSH scans. These had very different signatures from the other SSH scans and were non-malicious. The others are from the Cegeka dataset. The algorithm does peform quite well with an F-score of 0.76.

**Table 11.7:** K-Nearest Neighbors: real-world data

| | |
|---|---|
| F-score | 0.7633 |
| Total amount of samples | 11072646 |
| False negative | 8905 |
| False positive | 4164 |
| True negative | 11038482 |
| True positive | 21095 |

## 11.2   Decision Tree Classifier

Decision tree classifiers which are seen in Section 5.6 were thought of as not very promising since they work on a simple principle. The results do show that the algorithm is quite good at intrusion detection. The learning curve, shown in Figure 11.3 shows that the training score and the cross-validation score do converge towards eachother. However, they converge quite slowly. This means that the decision tree classifier exhibits high variance. Due to the fact that neither the training score or the cross-validation score suddenly drop, it can be concluded that there is no overfitting.
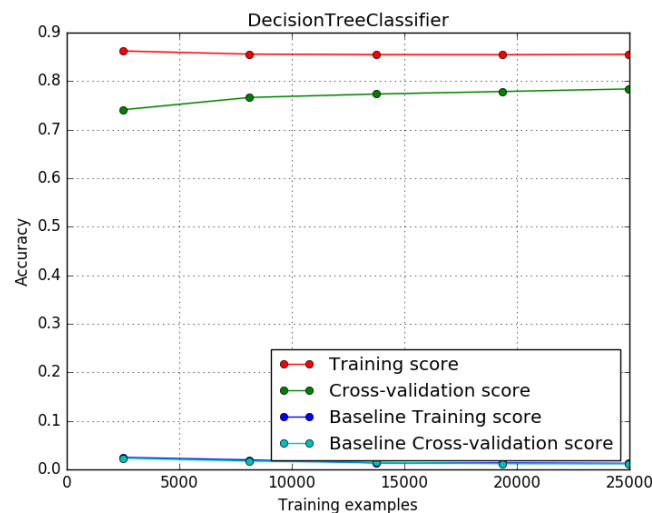


**Figure 11.3:** Learning Curve for Decision Tree Classifier

Table 11.8 shows the results from the experiments that were done on the CTU-dataset. It also shows the comparison between the different feature sets. The F-score for multi-class classification is lower than the F-score from KNN. The fact that this score is lower could be due to the same reason as to why Decision tree classifiers didn't seem to be a promising algorithm. The tree is constructed using "decisions". These "decisions" could be too simple to be able to accurately define a specific class.

However, the high binary F-score shows that even though the decisions might not always lead to the correct classification, they do lead to the correct decision of being malicious or non-malicious. The variance on the results is very small which means that the algorithm is stable and produces statistically relevant results.

The different features perform is a similar way as they performed for KNN. Using the TCP flags increases the F-scores of the algorithm. This means that somewhere in the tree, a decision is made using these TCP flags and that this decision does help with correctly classifying the samples. The country feature set does not increment the performance. The results between the country feature set and the standard feature set only differ with just a few samples. The decision tree classifier is not able to use the country of origin at all. This does seem to follow the assumption that the dataset just does not contain enough different countries to allow the algorithm to make any decisions.

**Table 11.8:** Decision Tree Classifier: comparing different feature sets: CTU-dataset

| Feature set | Standard | TCP | Country |
|---|---|---|---|
| Multi-class F-score | 0.6001 | 0.6650 | 0.5999 |
| Multi-class Precision | 0.6571 | 0.7099 | 0.6570 |
| Multi-class Recall | 0.6008 | 0.6733 | 0.6007 |
| Binary F-score | 0.9074 | 0.9511 | 0.9072 |
| Binary Precision | 0.8392 | 0.9175 | 0.8390 |
| Binary Recall | 0.9875 | 0.9873 | 0.9875 |
| Total amount of samples | 195519.0 | 195519.0 | 195519.0 |
| False negative | 708.9 | 722.5 | 710.5 |
| False positive | 10762.7 | 5050.6 | 10778.9 |
| True negative | 127865.3 | 133577.4 | 127849.1 |
| True positive | 56182.1 | 56168.5 | 56180.5 |
| Positive training samples | 25195.0 | 25195.0 | 25195.0 |
| Negative training samples | 24806.0 | 24806.0 | 24806.0 |
| Variance Multi-class F-score | 1.44e-07 | 1.58e-07 | 2.44e-07 |
| Variance Multi-class Precision | 1.85e-07 | 1.92e-07 | 1.73e-07 |
| Variance Multi-class Recall | 1.76e-07 | 1.79e-07 | 2.59e-07 |
| Variance Binary F-score | 2.92e-07 | 4.70e-08 | 5.80e-07 |
| Variance Binary Precision | 8.42e-07 | 9.15e-08 | 1.74e-06 |
| Variance Binary Recall | 8.92e-10 | 1.15e-07 | 1.56e-09 |

The results in Table 11.9, show the results of the cross-dataset evaluation which is used to evaluate the third step. Training is done using $19861$ non-malicious samples, $15000$ samples from the SQl dataset and $15000$ samples from the CTU dataset. The F-scores are higher than the F-scores from the CTU experiments. This means that the samples from the SQL dataset are more distinct and more easy to classify. It also means that the decision tree classifier is still able to correctly predict samples even though it is trained with multiple datasets.

**Table 11.9:** Decision Tree Classifier: comparing different feature sets: Cross-dataset

| Feature set | Standard | TCP | Country |
|---|---|---|---|
| Multi-class F-score | 0.7509 | 0.8042 | 0.7500 |
| Multi-class Precision | 0.7895 | 0.8307 | 0.7900 |
| Multi-class Recall | 0.7452 | 0.8057 | 0.7500 |
| Binary F-score | 0.9413 | 0.9716 | 0.9400 |
| Binary Precision | 0.8946 | 0.9506 | 0.8900 |
| Binary Recall | 0.9932 | 0.9935 | 0.9900 |
| Total amount of samples | 246467.0 | 246467.0 | 246467.0 |
| False negative | 729.7 | 700.9 | 723.9 |
| False positive | 12615.9 | 5567.7 | 12619.1 |
| True negative | 126012.1 | 133060.3 | 126008.9 |
| True positive | 107109.3 | 107138.1 | 107115.1 |
| Positive training samples | 30140.0 | 30140.0 | 30140.0 |
| Negative training samples | 19861.0 | 19861.0 | 19861.0 |
| Variance Multi-class F-score | 1.29e-07 | 1.93e-07 | 8.61e-08 |
| Variance Multi-class Precision | 1.98e-07 | 1.92e-07 | 1.08e-07 |
| Variance Multi-class Recall | 1.48e-07 | 3.21e-07 | 1.15e-07 |
| Variance Binary F-score | 7.93e-08 | 2.05e-07 | 9.44e-08 |
| Variance Binary Precision | 2.56e-07 | 2.59e-07 | 3.19e-07 |
| Variance Binary Recall | 6.43e-09 | 2.50e-07 | 4.49e-09 |

As a final step, Decision tree classifiers were evaluated using a real-life unlabeled dataset. This was done using the Cegeka and EDM dataset. In Table 11.7, the results can be seen. The algorithm was tested on $\sim 11000000$ samples. From these samples 10000 were from the EDM dataset. The others are from the Cegeka dataset. Surprisingly, the algorithm generated a huge amount of false positives. This caused to algorithm to have a really low F-score. Every sample from the EDM dataset was classified as malicious, while only 3 actually were. The reason that Decsion tree classifiers do not work well on real-world unlabeled data is probably because it is too fixated on the training data. The other evaluation tests got good results since the algorithm could work well on data that belonged to the same dataset. The decisions that the Decision tree classifier makes expect to be fed data that is very similar as the training dataset. This was the case for the other evaluation data, but not for the data from Cegeka and EDM.

**Table 11.10:** Decision tree classifiers: real-world data

| | |
|---|---|
| F-score | 0.0155 |
| Total amount of samples | 11072646 |
| False negative | 14280 |
| False positive | 3245349 |
| True negative | 7787297 |
| True positive | 25720 |

## 11.3 Naive Bayes

Naive Bayes has been explained in Section 5.7. The algorithm uses the Theorem of Bayes. Not every feature is independent, for example it could be that the source and destination port are connected. Due to this reason, the algorithm was not that promising. This also showed during the experiments.

The first experiment created the learning curve as seen in Figure 11.4. In the beginning, with just a few samples, the accuracy is higher than when more training samples are used. This is due to the distribution of labels. Some labels appear more than others. When only 500 samples are used for training, most of these samples belong to classes that appear the most. Due to the fact that there are not many training samples, the algorithm has a higher chance of only predicting the most popular classes. The distribution of labels is similar when more training samples are used, but due to the higher amount of training samples, the algorithm trains more, and accounts for more labels.

It can also be seen that both the training score and cross-validation score converge very fast. This signifies high bias and means that there is no point in training with more samples.
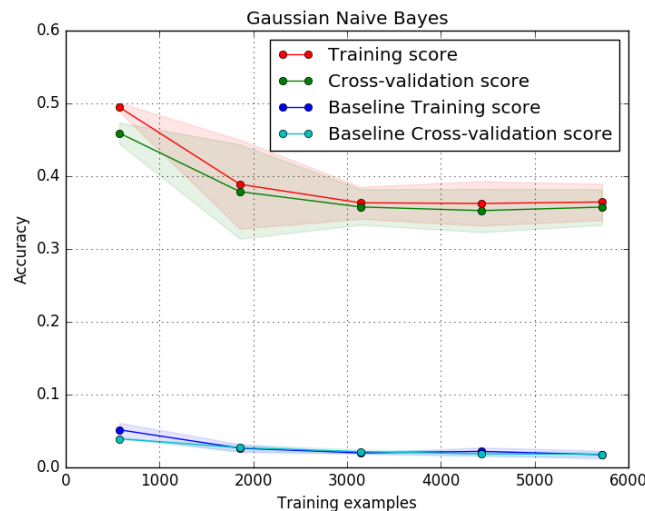


**Figure 11.4:** Learning Curve for Gaussian Naive Bayes

In Table 11.11, the results from the CTU experiments can be seen. Immediately two things can be seen. The scores are very low. The binary F-score is even lower than the F-score from the first baseline. The feature sets also have nearly no effect on the F-scores. The variance is also extremely small, which means these results are statistically relevant. This means that the Naive Bayes algorithm does not perform well at all. The fact that the feature sets have no effect can be contributed to the Naive assumption that is made.

If the features are dependent on eachother, then the Naive assumption is completely wrong. The features extracted from flows are dependent on eachother. This can be due to the fact that packets on a network are always structured. An example would be that when the destination port is 80, it can be expected that the protocol used is TCP. This signifies a dependence.

**Table 11.11:** Gaussian Naive Bayes: comparing different feature sets: CTU-dataset

| Feature set | Standard | TCP | Country |
|---|---|---|---|
| Multi-class F-score | 0.3253 | 0.3253 | 0.3253 |
| Multi-class Precision | 0.2980 | 0.2980 | 0.2980 |
| Multi-class Recall | 0.4231 | 0.4232 | 0.4232 |
| Binary F-score | 0.0146 | 0.0147 | 0.0147 |
| Binary Precision | 0.0179 | 0.0179 | 0.0179 |
| Binary Recall | 0.0123 | 0.0124 | 0.0124 |
| Total amount of samples | 195519.0 | 195519.0 | 195519.0 |
| False negative | 56191.2 | 56186.0 | 56186.0 |
| False positive | 38393.0 | 38495.0 | 38495.0 |
| True negative | 100235.0 | 100133.0 | 100133.0 |
| True positive | 699.8 | 705.0 | 705.0 |
| Positive training samples | 3545.0 | 3545.0 | 3545.0 |
| Negative training samples | 4956.0 | 4956.0 | 4956.0 |
| Variance Multi-class F-score | 3.08e-33 | 3.08e-33 | 3.08e-33 |
| Variance Multi-class Precision | 0.0 | 0.0 | 0.0 |
| Variance Multi-class Recall | 0.0 | 0.0 | 0.0 |
| Variance Binary F-score | 0.0 | 0.0 | 0.0 |
| Variance Binary Precision | 1.20e-35 | 1.20e-35 | 1.20e-35 |
| Variance Binary Recall | 0.0 | 0.0 | 0.0 |

During the cross-dataset validation in Table 11.12, similar results can be seen. It can be noted that the binary F-score is higher and barely above the binary F-score from the first baseline. This means that even if the features are not independent, the algorithm can still see differences between SSH scans and normal traffic. Due to the fact that Bayes does not perform well at all, it has not been used in step four.

**Table 11.12:** Gaussian Naive Bayes: comparing different feature sets: Cross-dataset

| Feature set | Standard | TCP | Country |
|---|---|---|---|
| Multi-class F-score | 0.3403 | 0.3398 | 0.3398 |
| Multi-class Precision | 0.3175 | 0.3170 | 0.3170 |
| Multi-class Recall | 0.4052 | 0.4047 | 0.4047 |
| Binary F-score | 0.4990 | 0.4988 | 0.4986 |
| Binary Precision | 0.4854 | 0.4847 | 0.4843 |
| Binary Recall | 0.5134 | 0.5136 | 0.5137 |
| Total amount of samples | 246467.0 | 246467.0 | 246467.0 |
| False negative | 52474.5 | 52452.9 | 52474.5 |
| False positive | 58695.1 | 58882.7 | 58695.1 |
| True negative | 79932.9 | 79745.3 | 79932.9 |
| True positive | 55364.5 | 55386.1 | 55364.5 |
| Positive training samples | 6545.0 | 6545.0 | 6545.0 |
| Negative training samples | 4956.0 | 4956.0 | 4956.0 |
| Variance Multi-class F-score | 3.26e-06 | 4.07e-06 | 4.92e-06 |
| Variance Multi-class Precision | 3.31e-06 | 3.80e-06 | 4.52e-06 |
| Variance Multi-class Recall | 2.72e-06 | 3.90e-06 | 3.95e-06 |
| Variance Binary F-score | 1.15e-06 | 1.31e-06 | 1.26e-06 |
| Variance Binary Precision | 5.09e-06 | 5.05e-06 | 3.07e-06 |
| Variance Binary Recall | 1.22e-06 | 8.18e-07 | 9.2e-07 |

## 11.4    Support Vector machines with Linear Kernel

Support Vector Machines have also been tested. These have been explained in Section 5.1. First Support Vector Machines with a linear kernel have been tested.  In the next section, Support Vector Machines with an RBF kernel has been used.

The learning curve, shown in Figure 11.5, is very irregular.  It does not seem to follow a pattern at all.  There is also a big standard deviation on the results.  The accuracy almost matches the baseline in the worst case.  Due to the irregular nature, it could lead to the conclusion that overfitting occurs since the cross-validation score drops.  However, the training score also drops, which is not expected during overfitting.  Underfitting could occur, but in that case it would be expected to see a slight increase in the accuracy instead on the extreme variance.
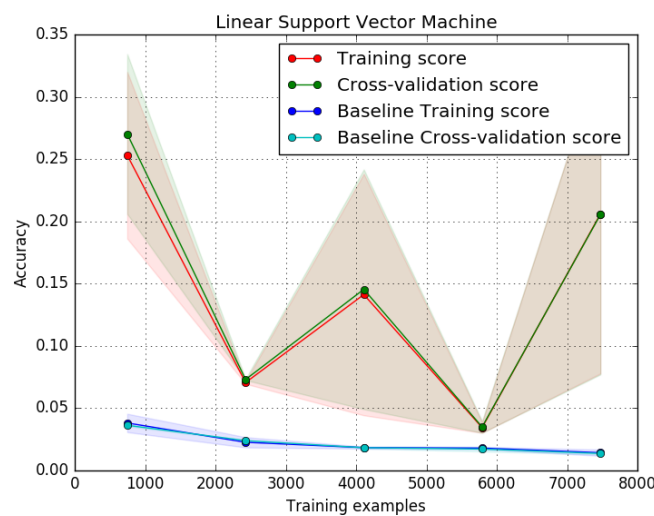


**Figure 11.5:** Learning Curve for Linear Support Vector Machine

In Table 11.13, the results from the experiments using the CTU dataset are shown.  The first element that can be seen is the high variance.  These mean that it becomes difficult to compare the results from the different feature sets.  Furthermore, the F-scores are barely equal to the F-scores from the baseline.

These scores and the variance can be explained due to the fact that intrusion detection cannot be solved using a linear classifier.  Since there are a lot of features, it is difficult to plot them, so Figure 5.2 from Section 5.1.1 will be used.  The linear classifier cannot group the different classes together.  Every group created by the linear classifier still contains a lot of samples from other classes.  This explains the low F-scores.  With multi-class classification, a different linear classifier is used for each class.  A linear classifier cannot find a good way to divide samples belonging to a class, and not belonging to the class.  In total, there are over a hunderd classes.  Each time the experiment is done, the linear classifier for each class is slightly different since it cannot find a way to correctly group each class.  This slight difference adds up for a hunderd classes and causes the high variance on the F-scores.

**Table 11.13:** Linear Kernel Support Vector Machine: comparing different feature sets: CTU-dataset

| Feature set | Standard | TCP | Country |
|---|---|---|---|
| Multi-class F-score | 0.1015 | 0.0191 | 0.1883 |
| Multi-class Precision | 0.2091 | 0.0437 | 0.3218 |
| Multi-class Recall | 0.0918 | 0.0330 | 0.1621 |
| Binary F-score | 0.4159 | 0.2267 | 0.4158 |
| Binary Precision | 0.2890 | 0.1470 | 0.2842 |
| Binary Recall | 0.7914 | 0.5007 | 0.7935 |
| Total amount of samples | 195519.0 | 195519.0 | 195519.0 |
| False negative | 11866.5 | 28402.9 | 11747.6 |
| False positive | 90404.1 | 76436.5 | 87959.7 |
| True negative | 48223.9 | 62191.5 | 50668.3 |
| True positive | 45024.5 | 28488.1 | 45143.4 |
| Positive training samples | 10069.0 | 10069.0 | 10069.0 |
| Negative training samples | 9932.0 | 9932.0 | 9932.0 |
| Variance Multi-class F-score | 0.02 | 0.05 | 0.02020 |
| Variance Multi-class Precision | 0.03 | 0.02 | 0.0279 |
| Variance Multi-class Recall | 0.02 | 0.0001 | 0.0189 |
| Variance Binary F-score | 0.05 | 0.05 | 0.0464 |
| Variance Binary Precision | 0.03 | 0.02 | 0.0233 |
| Variance Binary Recall | 0.15 | 0.25 | 0.1574 |

The cross-dataset experiments shown in Table 11.14 have a lower variance. However, compared to other algorithms, the variance is still very high. The F-scores are overall a bit higher than the F-scores from the baseline, which means that the F-scores are not great at all. The results from this experiment are similar to the CTU experiment and the explanation of why the algorithm does not perform well is still supported due to the low scores and the high variance. Due to the fact that Support Vector Machines using a linear kernel do not perform well at all, it has not been used in step four.

**Table 11.14:** Linear Kernel Support Vector Machine: comparing different feature sets: Cross-dataset

| Feature set | Standard | TCP | Country |
|---|---|---|---|
| Multi-class F-score | 0.2009 | 0.0369 | 0.2095 |
| Multi-class Precision | 0.3925 | 0.0533 | 0.4191 |
| Multi-class Recall | 0.1478 | 0.0497 | 0.1556 |
| Binary F-score | 0.6041 | 0.4974 | 0.6390 |
| Binary Precision | 0.4952 | 0.4263 | 0.4703 |
| Binary Recall | 0.9183 | 0.6796 | 0.9971 |
| Total amount of samples | 246467.0 | 246467.0 | 246467.0 |
| False negative | 8806.2 | 34551.5 | 314.2 |
| False positive | 112296.5 | 89987.9 | 121331.3 |
| True negative | 26331.5 | 48640.1 | 517296.7 |
| True positive | 99032.8 | 73287.5 | 107524.8 |
| Positive training samples | 10045.0 | 10045.0 | 10045.0 |
| Negative training samples | 4956.0 | 4956.0 | 4956.0 |
| Variance Multi-class F-score | 0.0023 | 0.0027 | 0.0009 |
| Variance Multi-class Precision | 0.0041 | 0.0053 | 0.0004 |
| Variance Multi-class Recall | 0.0022 | 0.0049 | 0.0009 |
| Variance Binary F-score | 0.0084 | 0.0309 | 0.0001 |
| Variance Binary Precision | 0.0084 | 0.0240 | 0.0002 |
| Variance Binary Recall | 0.0551 | 0.1003 | 1.1332e-06 |

## 11.5 Support Vector machines with RBF Kernel

Support Vector Machines using a linear kernel did not work at all. They were barely better than the baseline. A conclusion was made that the classification cannot be made using linear classification. This can be verified by using a non-linear kernel. The kernel that was used, is the RBF kernel as seen in Section 5.1.2.

The learning curve is shown in Figure 11.6. Immediatly can be seen that the learning curve looks much better. The curve is regular. In this learning curve, it can be seen that there is a big gap between the cross-validation and training score. In underfitting, both the training and cross-validation score should be low. This cannot be seen in this learning curve. During overfitting, it would be expected that the training score is high since the model fits the training data very well, however the cross-validation score should be low and dropping. However, this also cannot be seen in this learning curve.
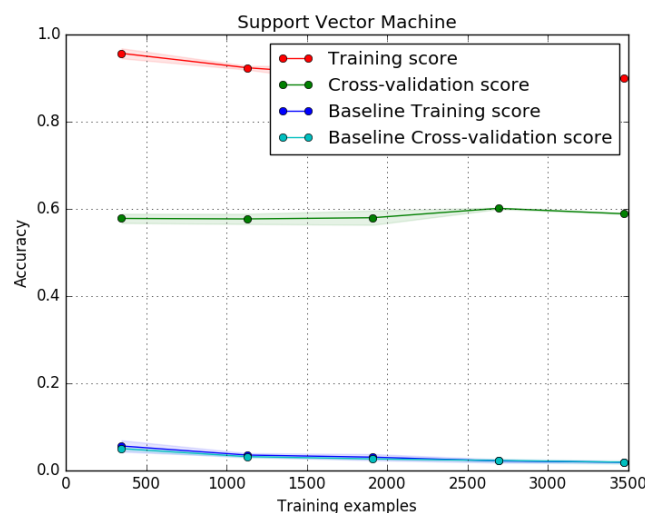


**Figure 11.6:** Learning Curve for Support Vector Machine

The results from the CTU experiments are shown in Table 11.15. The variance on both F-scores are very low which means we can compare the different F-scores. The multi-class F-scores are higher than the multi-class F-scores from the experiments with the linear kernel. This means that the exact classification happens better when using a non-linear kernel. This supports the idea that the classification cannot happen with a linear classifier.

However, the binary F-score is extremely low. This is a result from using the RBF kernel. RBF kernels are also called Gaussian kernels. The kernel expects that training data follows a Gaussian distribution which is not the case for the training data from intrusion detection. The malicious and non-malicious samples do not follow a Gaussian distribution at all. When trying to classify samples using a Gaussian distribution, it cannot see the difference between malicious or non-malicious. More concretely, it seems that the algorithm cannot detect malicious samples at all. It classifies most samples as non-malicious. However, it can still see the difference between some non-malicious classes. Because of that the multi-class F-score seems to be higher.

Even though the binary F-score is much lower than the binary F-score from the experiments with linear kernels, both are worse than the baseline. The results from the multi-class F-score can also be compared to the multi-class F-score from the linear kernels. The multi-class F-score is still much higher than the multi-class F-score from the baseline. This means that the algorithm can still classify results better than

randomly. The different feature sets also have little to no effect on the F-score, in case of the TCP feature set, it even lowers the F-score.

**Table 11.15:** RBF Kernel Support Vector Machine: comparing different feature sets: CTU-dataset

| Feature set | Standard | TCP | Country |
|---|---|---|---|
| Multi-class F-score | 0.3641 | 0.3030 | 0.3643 |
| Multi-class Precision | 0.5124 | 0.5554 | 0.5167 |
| Multi-class Recall | 0.4037 | 0.3589 | 0.4062 |
| Binary F-score | 0.0455 | 0.0282 | 0.0449 |
| Binary Precision | 0.0749 | 0.9987 | 0.0758 |
| Binary Recall | 0.0327 | 0.0143 | 0.0319 |
| Total amount of samples | 195519.0 | 195519.0 | 195519.0 |
| False negative | 55029.0 | 56077.5 | 55075.0 |
| False positive | 22991.0 | 1.0 | 22115.0 |
| True negative | 115637.0 | 138627.0 | 116513.0 |
| True positive | 1862.0 | 813.5 | 1816.0 |
| Positive training samples | 2522.0 | 2522.0 | 2522.0 |
| Negative training samples | 2479.0 | 2479.0 | 2479.0 |
| Variance Multi-class F-score | 3.08e-33 | 0.0 | 3.08e-33 |
| Variance Multi-class Precision | 0.0 | 1.23e-32 | 1.23e-32 |
| Variance Multi-class Recall | 1.23e-32 | 0.0 | 3.08e-33 |
| Variance Binary F-score | 0.0 | 1.20e-35 | 4.81e-35 |
| Variance Binary Precision | 0.0 | 1.23e-32 | 0.0 |
| Variance Binary Recall | 4.81e-35 | 0.0 | 4.81e-35 |

The results from the cross-dataset validation are shown in Table 11.16. These results are much better. It seems that the SSH scans can be detected better than the malware from the CTU dataset. The effect of using the TCP feature set and Country feature set is quite significant. This means that these feature do help the algorithm to correctly classifiy samples. However, this seems to only be the case for the SSH scans, not the samples from the CTU dataset.

This can have two explanations. The first explanation is that using the TCP flags that are used allow the SVM to make a distinction between samples from the SQL dataset and the CTU dataset. This means that during predictions, it can also distinguish between samples from the SQL dataset and the CTU dataset. In other words, the algorithm is affected by the origin of the data.

The other explanation is that the distinction between SSH scans and normal behaviour is quite large when the TCP flags are used, but malware and normal behaviour still look the same to the SVM. This means that the SVM can see differences between some classes, but not between all classes. Since such a difference between CTU datasets and Cross-dataset experiments can only be seen in experiments using SVM with a RBF kernel, it seems that the second explanation is more plausible. Due to the fact that Support Vector Machines using a RBF kernel do not perform well at all, it has not been used in step four.

**Table 11.16:** RBF Kernel Support Vector Machine: comparing different feature sets: Cross-dataset

| Feature set | Standard | TCP | Country |
|---|---|---|---|
| Multi-class F-score | 0.3352 | 0.5761 | 0.5764 |
| Multi-class Precision | 0.6524 | 0.6944 | 0.6982 |
| Multi-class Recall | 0.4277 | 0.5979 | 0.5997 |
| Binary F-score | 0.6740 | 0.6974 | 0.7001 |
| Binary Precision | 0.5083 | 0.7492 | 0.7567 |
| Binary Recall | 0.9999 | 0.6524 | 0.6514 |
| Total amount of samples | 246467.0 | 246467.0 | 246467.0 |
| False negative | 10.8 | 37484.0 | 37584.4 |
| False positive | 104306.8 | 23541.0 | 22587.7 |
| True negative | 34321.2 | 115087.0 | 116040.3 |
| True positive | 107828.2 | 70355.0 | 70254.6 |
| Positive training samples | 4522.0 | 4522.0 | 4522.0 |
| Negative training samples | 2479.0 | 2479.0 | 2479.0 |
| Variance Multi-class F-score | 4.46e-08 | 3.92e-08 | 2.64e-07 |
| Variance Multi-class Precision | 6.53e-06 | 3.30e-08 | 8.75e-08 |
| Variance Multi-class Recall | 2.16e-08 | 1.00e-07 | 6.01e-07 |
| Variance Binary F-score | 5.71e-09 | 2.65e-07 | 1.42e-06 |
| Variance Binary Precision | 6.50e-09 | 8.58e-08 | 1.96e-07 |
| Variance Binary Recall | 5.50e-09 | 4.76e-07 | 3.0685e-06 |

## 11.6   Neural network

Neural networks were a promising algorithm. They were explained in Section 5.5. Neural networks are quite complex and contain a lot of weights that need to be calculated during the training phase. Because of this, they require to be trained using a lot of data or they can be unstable and be in an underfitting phase.

The learning curve can be seen in Figure 11.7. The first thing that can be seen is that the training score and the cross-validation score are very close together. The scores are also dropping. Neural network requires a lot of training data. However, in the available datasets there is only a couple of thousand samples per malicious classification. The accuracy keeps dropping. This does not signify overfitting as the training score is also low. However, it does seem to be related to underfitting. During underfitting, it is expected that the training and cross-validation score drop to a local minimum and afterwards starts increases as the overfitting is resolved. However, there is not enough data to test when the overfitting is resolved.
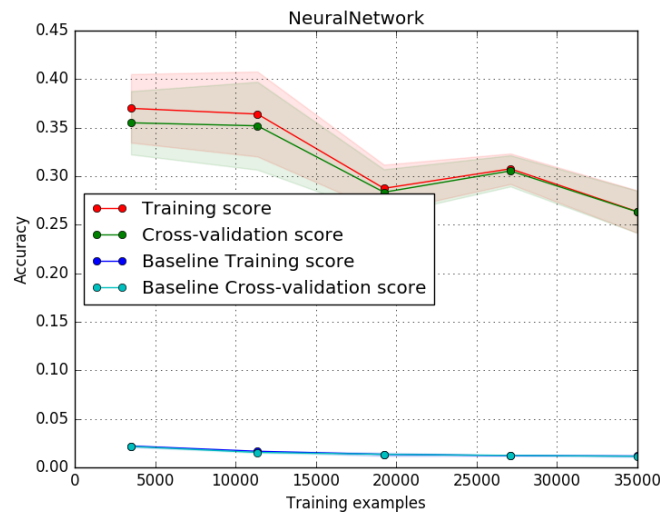


**Figure 11.7:** Learning Curve for Neural Network

Even though the algorithm underfits, the experiments were still done. The results from the CTU dataset are shown in Table 11.17. The Binary F-score is 0 which means that the algorithm always classifies samples as non-malicious. The multi-class F-score is also quite low. This could be contributed to the underfitting.

The biggest piece of evidence is the F-scores for the different feature sets. The multi-class F-scores for the TCP feature set and the Country feature set are much lower. These feature sets contain more features compared to the standard feature set. In a neural network, this means that more weights need to be calculated. This means that the overfitting should be worse when using more features. This is reflected in the F-scores for the TCP and Country feature set.

**Table 11.17:** Neural Network: comparing different feature sets: CTU-dataset

| Feature set | Standard | TCP | Country |
|---|---|---|---|
| Multi-class F-score | 0.2213 | 0.1322 | 0.1763 |
| Multi-class Precision | 0.2070 | 0.0854 | 0.1479 |
| Multi-class Recall | 0.3541 | 0.2922 | 0.3225 |
| Binary F-score | 0.0 | 0.0 | 0.0 |
| Binary Precision | 0.0 | 0.0 | 0.0 |
| Binary Recall | 0.0 | 0.0 | 0.0 |
| Total amount of samples | 195519.0 | 195519.0 | 195519.0 |
| False negative | 56891.0 | 56891.0 | 56891.0 |
| False positive | 0.0 | 0.0 | 0.0 |
| True negative | 138628.0 | 138628.0 | 138628.0 |
| True positive | 0.0 | 0.0 | 0.0 |
| Positive training samples | 25195.0 | 25195.0 | 25195.0 |
| Negative training samples | 24806.0 | 24806.0 | 24806.0 |
| Variance Multi-class F-score | 0.0019 | 7.70e-34 | 0.0029 |
| Variance Multi-class Precision | 0.0037 | 0.0 | 0.0059 |
| Variance Multi-class Recall | 0.0009 | 0.0 | 0.0013 |
| Variance Binary F-score | 0.0 | 0.0 | 0.0 |
| Variance Binary Precision | 0.0 | 0.0 | 0.0 |
| Variance Binary Recall | 0.0 | 0.0 | 0.0 |

The results from the cross-dataset validation is shown in Table 11.18. The scores are yet again, quite low. The algorithm was trained using 20000 samples from the SQL dataset, 25000 malicious samples from the CTU dataset and 25000 non-malicious samples from the CTU dataset.

The binary F-score is much higher as compared to the CTU experiment. It has to be noted that the samples of the SQL dataset belong to a single class, SSH scans and that the malicious samples from the CTU dataset contain a lot of different malicious classes. This means that if the binary F-score is higher, the neural net does not underfit that much anymore on the classification between malicious and non-malicious. This is due to the SSH scan samples. This leads to the conclusion that the algorithm is should not underfit that much anymore when more than 20000 samples are used for each class.

It is also interesting that the multi-class F-score for the TCP feature set is much higher as compared to the other multi-class F-scores. This is also due to the samples from the SQL dataset. It seems that having these samples does indeed resolve a part of the underfitting, especially when the TCP flags are used. Even though more weights need to be calculated, it seems that the TCP flags help more than they cause underfitting. Due to the fact that Neural networks do not perform well and are underfitting, it has not been used in step four.

**Table 11.18:** Neural Network: comparing different feature sets: Cross-dataset

| Feature set | Standard | TCP | Country |
|---|---|---|---|
| Multi-class F-score | 0.1338 | 0.3679 | 0.1433 |
| Multi-class Precision | 0.0908 | 0.3328 | 0.0968 |
| Multi-class Recall | 0.2901 | 0.5021 | 0.2986 |
| Binary F-score | 0.6128 | 0.5514 | 0.6056 |
| Binary Precision | 0.4426 | 0.5507 | 0.4437 |
| Binary Recall | 0.9970 | 0.5635 | 0.9641 |
| Total amount of samples | 246467.0 | 246467.0 | 246467.0 |
| False negative | 316.9 | 47071.7 | 3867.5 |
| False positive | 135732.8 | 49578.2 | 130776.2 |
| True negative | 2895.2 | 89049.8 | 7851.8 |
| True positive | 107522.1 | 60767.3 | 103971.5 |
| Positive training samples | 45195.0 | 45195.0 | 45195.0 |
| Negative training samples | 24806.0 | 24806.0 | 24806.0 |
| Variance Multi-class F-score | 0.0006 | 0.0019 | 0.0013 |
| Variance Multi-class Precision | 0.0009 | 0.0028 | 0.0010 |
| Variance Multi-class Recall | 0.0003 | 0.0017 | 0.0009 |
| Variance Binary F-score | 0.0001 | 0.0024 | 0.0006 |
| Variance Binary Precision | 0.0002 | 0.0039 | 0.0002 |
| Variance Binary Recall | 7.77e-05 | 0.0075 | 0.009 |

## 11.7 One-class Support Vector Machines

One-class Support Vector Machines are the only unsupervised learning algorithms that have been used. They are explained in Section 5.4. One-class Support Vector Machines only use a single class. The idea behind the algorithm was to perhaps use it as a preprocessor. If the algorithm would be able to correctly predict whether an element is malicious or non-malicious, it could be used before a supervised algorithm would be used. Afterwards, only the malicious samples would be fed to a supervised learning algorithm.

Table 11.19 shows the results from different experiments that have been done. Yet again, it can be seen that using the TCP feature set does increase the performance. However, even with the TCP feature set, the binary F-score is barely as good as the baseline, a random classifier. Looking more closely, it can be seen that there are no true negatives in the TCP feature set experiment. However, there are false negatives. This means that the algorithm cannot be used at all for a preprocessing step.

As said before, one-class Support Vector Machine uses only a single class. Once they are trained, they can predict whether an element belongs to this class or not. This class is made by using a RBF kernel. The results show that the SVM cannot properly define the boundaries of the class. This means that there are a lot of samples that are predicted wrong. Using the TCP flags probably make the class boundary even more irregular as compared to using the standard feature set. This means that the class becomes so large that almost every possible sample becomes part of the class. This causes the effect that even the non-malicious samples are all predicted to be malicious. This effect is not seen in the country feature set. That can be explained by the fact that in the country feature set, the IP address features are replaced by the country of origin. This has the effect that the class is defined more or less the same.

**Table 11.19:** One-class Support Vector Machine: comparing different feature sets: Cross-dataset

| Feature set | Standard | TCP | Country |
|---|---|---|---|
| Binary F-score | 0.1275 | 0.4017 | 0.1557 |
| Binary Precision | 0.1128 | 0.2911 | 0.1385 |
| Binary Recall | 0.1489 | 0.6480 | 0.1794 |
| Total amount of samples | 226467.0 | 226467.0 | 226467.0 |
| False negative | 74762.4 | 30912.0 | 72082.7 |
| False positive | 88162.5 | 138628.0 | 87260.7 |
| True negative | 50465.6 | 0.0 | 51367.3 |
| True positive | 13076.5 | 56927.0 | 15756.3 |
| Positive training samples | 65000.0 | 65000.0 | 65000.0 |
| Negative training samples | 0.0 | 0.0 | 0.0 |
| Variance Binary F-score | 0.0191 | 0.0006 | 0.0169 |
| Variance Binary Precision | 0.0125 | 0.0002 | 0.0116 |
| Variance Binary Recall | 0.0319 | 0.0024 | 0.0261 |

# Chapter 12

# Future work

## 12.1 Increasing performance

There are several possible methods that could be used to improve the performance of the implemented system. Currently the system generates alerts based on whether it thinks a flow is malicious or not. Incrementing the performance of the system could be solved by giving the system more data to learn from.

One method to be able to get more training data is to generate it. This would be very useful for the neural network algorithm which was underfitting. Using a Virtual Machine, different kinds of malware could be run. The generated network traffic can be used to train the system.

Another possible solution could be to implement packet analysis. A flow-based intrusion detection system processes a lot less data than a packet-based intrusion detection system. However, because IP flows contain less information than complete packets, a flow-based system cannot detect everything. It might be worth to find methods that incorperate both flow-based intrusion detection and packet-based intrusion deteciton.

One such method could be to use the IP flows to eliminate flows that are non-malicious and afterwards to perform packet analysis on the remaining flows. This may or may not have an impact on the accuracy and the different kinds of attacks that can be detected.

## 12.2 Intrusion Prevention Systems

It would be interesting to find out whether the methods described in this thesis also apply to intrusion prevention systems. There are two main issues with using a similar system for intrusion prevention. IP Flows are defined as an entire "flow", as a connection. At the moment a flow is received by the intrusion detection system, every packet has already been exchanged between the source and the destination. This means that "prevention" is more difficult since the intrusion has already happened. A solution to this could be to process incomplete flows. This means that the flow might still change and that communication is still ongoing. However, it is unknown how the system reacts to unknown flows.

The other issue is the performance of the machine learning algorithms. During the tests that were done in this thesis, no performance issues of the machine learning algorithms were noticed. But it is unknown how fast the machine learning algorithm needs to perform, should the intrusion detection system be fast enough.

## 12.3 Combination of algorithms

In this thesis, multiple algorithms have been used. However, these algorithms were tested separately. It would be interesting to find out whether these algorithms can be combined. This could be done by training multiple algorithms and using the results in such a way that the multi-class and binary F-scores

are maximised. One method of doing this could be by minimizing the amount of false positives and false negatives.

Another method is to use a layered approach. In this method, algorithms are chained together in a layered method. Every layer reduces the amount of flows that the next layer should process. This approach works in a similar way as explained in Section 10.5. However, that approach only speaks about using an unsupervised learning algorithm and afterwards a supervised learning algorithm.

## 12.4 Using only binary classification

This thesis focused on multi-class classification. The different machine learning algorithms are trained in order to be able to predict the exact classification of a flow. The binary F-score that was used in the evaluation was calculated after the algorithms predicted the classes. It would be interesting to see what kind of results can be obtained by training the machine learning algorithms in a binary way. This means that the algorithm can only predict whether a flow is malicious or non-malicious.

The framework that was made for this thesis does already support binary learning. In Table 12.1 and Table 12.2, some results can be seen. These experiments have only been repeated a couple times and are not necessarially statistically correct. However, they do give an indication whether binary classification offers an advantage. The results show excellent binary F-scores which are better than the binary scores shown in the binary F-score for multiclass Classification.

**Table 12.1:** K-Nearest Neighbors (standard feature set, k=5) for CTU Dataset

| Experiment | multi-class training | binary training |
| --- | --- | --- |
| Binary F-score | 0.9500 | 0.9859 |
| Binary Precision | 0.9204 | 0.9803 |
| Binary Recall | 0.9816 | 0.9917 |
| Total amount of samples | 195519.0 | 195519.0 |
| False negative | 1048.0 | 469.0 |
| False positive | 4830.0 | 1136.0 |
| True negative | 133798.0 | 137492.0 |
| True positive | 55843.0 | 56422.0 |
| Positive training samples | 35363.0 | 35363.0 |
| Negative training samples | 49638.0 | 49638.0 |

**Table 12.2:** K-Nearest Neighbors (standard feature set, k=5) for Cross Dataset

| Experiment | multi-class training | binary training |
| --- | --- | --- |
| Binary F-score | 0.9631 | 0.9926 |
| Binary Precision | 0.9543 | 0.9895 |
| Binary Recall | 0.9720 | 0.9957 |
| Total amount of samples | 246467.0 | 246467.0 |
| False negative | 3015.1 | 468.0 |
| False positive | 5013.9 | 1137.0 |
| True negative | 133614.1 | 137491.0 |
| True positive | 104823.9 | 107371.0 |
| Positive training samples | 65363.0 | 65363.0 |
| Negative training samples | 49638.0 | 49638.0 |

# Chapter 13

# Conclusion

This thesis has given an overview of machine learning algorithms and has shown how they can be used in an intrusion detection system. The advantages and disadvantages of IP flows have been given. It was found that IP flows do contain enough information to detect intrusions.

However, they do not contain enough information to detect all different types of intrusions. They can only detect a subset of intrusions. In general, the more information is known, the better the results. Extra information such as knowing the TCP flags used in the flow can greatly improve the performance of machine learning algorithms. Other information such as using the country of origin of a flow did not have any impact on the results. However, this could be caused by a lack of variance in the training dataset. The experiments show that attacks such as SSH scans are more easily detectable as compared to malware.

The biggest problem that was discovered during the thesis was finding good labeled datasets which could be used to train the machine learning algorithms. If a good training dataset is used to train a machine learning algorithm, it can be used to create an intrusion detection system which offers acceptable performance out-of-the-box. A lot depends on the quality of the training dataset. If the training dataset does not contain enough samples of the different intrusions, the machine learning algorithm will exhibit a large amount of false positives and false negatives.

Not all machine learning algorithms work as good. The Naive Bayes algorithm makes wrong assumptions about the features. This makes it a poor choice for intrusion detection. Neural networks seem promising but there was not enough data to train the algorithm. Most experiments with neural networks were done while it was underfitting. Decision tree algorithms have promising results in the experiments, however they do not hold up when they are used in a real-life scenario.

Support Vector Machines can be used with a linear kernel or a RBF kernel. The linear kernel SVM does not work at all, since intrusion detection is not a problem that can be solved with linear classifiers. Support vector machines with a RBF kernel perform much better. However, there performance is still quite average compared to the best algorithm. K-Nearest Neighbors performed the best. It has good results in both the evaluation and the real-life scenario.

When using an algorithm such as K-Nearest Neighbors close attention needs to be paid to what value of $k$ is chosen and which distance metric is used. The value of $k$ needs to be chosen experimentally, since the results in this thesis show that there is not a best value of $k$. The Canberra distance metric works the best for multi-class classification. Otherwise, if there is only an interest towards binary classification, the Euclidean or Chebyshev distance metric should be chosen.

Unsupervised learning algorithms do not work well out-of-the-box. They need a lot of manual interference before they are viable to be used for intrusion detection.

# Bibliography

[1] *Hospital declares 'internal state of emergency' after ransomware infection*, http://krebsonsecurity.com/2016/03/hospital-declares-internet-state-of-emergency-after-ransomware-infection/, Accessed: 2016-05-12.

[2] *Sans: Host vs network based intrusion detection systems*, https://cyber-defense.sans.org/resources/papers/gsec/host-vs-network-based-intrusion-detection-systems-102574, Accessed: 2016-05-11.

[3] M. J. Martin, *Smart grid: Cyber security*, https://www.linkedin.com/pulse/smart-grid-cyber-security-martin-ma-mba-med-gdm-scpm-pmp, Accessed: 2016-03-24.

[4] R. G. Bace, *Intrusion Detection*. Indianapolis, IN, USA: Macmillan Publishing Co., Inc., 2000, ISBN: 1-57870-185-6.

[5] A. Sundaram, "An introduction to intrusion detection", *Crossroads*, vol. 2, no. 4, pp. 3–7, Apr. 1996, ISSN: 1528-4972. DOI: 10.1145/332159.332161. [Online]. Available: http://doi.acm.org/10.1145/332159.332161.

[6] H. Alaidaros, M. Mahmuddin, and A. Al Mazari, "An overview of flow-based and packet-based intrusion detection performance in high speed networks", in *Proceedings of the International Arab Conference on Information Technology*, 2011.

[7] T.-L. Pao and P.-W. Wang, "Netflow based intrusion detection system", in *Networking, Sensing and Control, 2004 IEEE International Conference*, IEEE, vol. 2, 2004, pp. 731–736.

[8] *Sans: Understanding ips and ids: Using ips and ids together for defense in depth*, https://www.sans.org/reading-room/whitepapers/detection/understanding-ips-ids-ips-ids-defense-in-depth-1381, Accessed: 2016-05-11.

[9] M. Golling, R. Hofstede, and R. Koch, "Towards multi-layered intrusion detection in high-speed networks", in *Cyber Conflict (CyCon 2014), 2014 6th International Conference*, IEEE, 2014, pp. 191–206.

[10] *Ips [intrusion prevention systems] – your 2nd line of defense*, https://bastionnux.wordpress.com/2011/04/06/ips-intrustion-prevention-systems-your-2nd-line-of-defense/, Accessed: 2016-05-11.

[11] *Elprocus: Basic intrusion detection system*, https://www.elprocus.com/basic-intrusion-detection-system/, Accessed: 2016-05-12.

[12] M. J. N. Jayveer Singh, "A survey on machine learning techniques for intrusion detection systems", *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 2, no. 11, 2013.

[13] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An overview of ip flow-based intrusion detection.", *IEEE Communications Surveys and Tutorials*, vol. 12, no. 3, pp. 343–356, 2010. [Online]. Available: http://dblp.uni-trier.de/db/journals/comsur/comsur12.html#SperottoSSMPS10.

[14] M. Uddin, R. Alsaqour, and M. Abdelhaq, "Intrusion detection system to detect ddos attack in gnutella hybrid p2p network", *Indian Journal of Science and Technology*, vol. 6, no. 2, pp. 4045–4057, 2013.

[15] *Alienvault: Who we are*, https://www.alienvault.com/who-we-are, Accessed: 2016-05-11.

[16] *Alienvault: Products*, https://www.alienvault.com/products, Accessed: 2016-05-11.

[17]   *Alienvault: Intrusion detection techniques: Methods and best practices*, `https://www.alienvault.com/blogs/security-essentials/intrusion-detection-techniques-methods-best-practices`, Accessed: 2016-05-11.

[18]   G. Kurundkar, N. Naik, and S. Khamitkar, "Network intrusion detection using snort", *International Journal of Engineering Research and Applications*, vol. 2, no. 2, pp. 1288–1296, 2012.

[19]   N. Duffield, P. Haffner, B. Krishnamurthy, and H. Ringberg, "Rule-based anomaly detection on ip flows", in *INFOCOM 2009, IEEE*, IEEE, 2009, pp. 424–432.

[20]   K. Veeramachaneni and I. Arnaldo, "Ai2: Training a big data machine to defend",

[21]   *Mit designed ai2, the system that can detect 85attacks*, `http://securityaffairs.co/wordpress/46480/security/ai2-system.html`, Accessed: 2016-05-12.

[22]   N Paulauskas and E Garsva, "Computer system attack classification", *Elektronika ir Elektrotechnika*, vol. 66, no. 2, pp. 84–87, 2015.

[23]   S. Hansman and R. Hunt, "A taxonomy of network and computer attacks", *Computers & Security*, vol. 24, no. 1, pp. 31–43, 2005.

[24]   *Man in the middle part 1: Introduction*, `https://toschprod.wordpress.com/2011/11/06/man-in-the-middle-intro/`, Accessed: 2016-04-28.

[25]   *A peek at the future of botnet evolution*, `http://securitywatch.pcmag.com/none/309491-a-peek-at-the-future-of-botnet-evolution`, Accessed: 2016-04-28.

[26]   A.-S. Kim, H.-J. Kong, S.-C. Hong, S.-H. Chung, and J. W. Hong, "A flow-based method for abnormal network traffic detection", in *Network operations and management symposium, 2004. NOMS 2004. IEEE/IFIP*, IEEE, vol. 1, 2004, pp. 599–612.

[27]   D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the slammer worm", *IEEE Security & Privacy*, no. 4, pp. 33–39, 2003.

[28]   Y. Abuadlla, G. Kvascev, S. Gajin, and Z. Jovanovic, "Flow-based anomaly intrusion detection system using two neural network stages", *Computer Science and Information Systems*, vol. 11, no. 2, pp. 601–622, 2014.

[29]   Z. Zhu, G. Lu, Y. Chen, Z. J. Fu, P. Roberts, and K. Han, "Botnet research survey", in *Computer Software and Applications, 2008. COMPSAC'08. 32nd Annual IEEE International*, IEEE, 2008, pp. 967–972.

[30]   *Coursera machine learning stanford university*, `https://www.coursera.org/learn/machine-learning`, Accessed: 2016-02-09.

[31]   *Coursera machine learning lectures stanford university*, `https://class.coursera.org/ml-003/lecture`, Accessed: 2016-02-09.

[32]   *scikit-learn linear regression*, `http://scikit-learn.org/stable/modules/linear_model.html`, Accessed: 2016-02-15.

[33]   *Stanford cs229 machine learning*, `http://cs229.stanford.edu/materials.html`, Accessed: 2016-04-18.

[34]   *Intuitive machine learning : Gradient descent simplified*, `http://ucanalytics.com/blogs/intuitive-machine-learning-gradient-descent-simplified/`, Accessed: 2016-04-20.

[35]   *Stackoverflow gradient descent convergence how to decide convergence?*, `http://stackoverflow.com/questions/17289082/gradient-descent-convergence-how-to-decide-convergence`, Accessed: 2016-04-18.

[36]   *Coursera machine learning gradient descent stanford university*, `https://class.coursera.org/ml-003/lecture/11`, Accessed: 2016-02-09.

[37]   *Cis 520 machine learning - 2015*, `https://alliance.seas.upenn.edu/~cis520/dynamic/2014/wiki/index.php?n=Lectures.Classification`, Accessed: 2016-04-22.

[38] *Coursera machine learning cost function stanford university*, https://class.coursera.org/ml-003/lecture/6, Accessed: 2016-02-09.

[39] C. M. Bishop, "Pattern recognition and machine learning. springer".

[40] *Logistic regression*, http://www.holehouse.org/mlclass/06_Logistic_Regression.html, Accessed: 2016-04-22.

[41] *Test-Driven Data Analysis in defence of xml: Exporting and analysing apple health data*, http://www.tdda.info/, Accessed: 2016-04-18.

[42] *Wolfram mathworld: Distance*, http://mathworld.wolfram.com/Distance.html, Accessed: 2016-04-24.

[43] *Creating my first algorithm from scratch – euclidean distance and pearson correlation*, https://bigsnarf.wordpress.com/2012/03/25/creating-my-first-algorithm-from-scratch-euclidean-distance-and-pearson-correlation/, Accessed: 2016-04-24.

[44] *Hamming distance*, https://people.rit.edu/rmb5229/320/project3/hamming.html, Accessed: 2016-04-24.

[45] *Chebyshev distance explaination*, https://chessprogramming.wikispaces.com/Distance, Accessed: 2016-04-25.

[46] *Chebyshev distance*, https://en.wikipedia.org/wiki/Chebyshev_distance, Accessed: 2016-04-25.

[47] *Privacy preserving scheme for location-based services*, http://file.scirp.org/Html/6-7800068_18792.htm, Accessed: 2016-04-24.

[48] R. V. Giuseppe Jurman Samantha Riccadonna and C. Furlanello, "Canberra distance on ranked lists", Accessed: 2016-04-24. [Online]. Available: \url{http://mpba.fbk.eu/files/publications/jurman09canberra.pdf}.

[49] *The minkowski distance (lp)*, https://renatocamorim.com/2012/09/20/calculating-the-minkowski-center-l_p/, Accessed: 2016-04-25.

[50] *Simplification and shift in cognition of political difference: Applying the geometric modeling to the analysis of semantic similarity judgment.* https://openi.nlm.nih.gov/detailedresult.php?img=PMC3108968_pone.0020693.g001&req=4, Accessed: 2016-04-25.

[51] *DTreg svm - support vector machines*, https://www.dtreg.com/solution/view/20, Accessed: 2016-04-18.

[52] *Why is logistic regression a linear classifier?*, http://stats.stackexchange.com/questions/93569/why-is-logistic-regression-a-linear-classifier, Accessed: 2016-04-24.

[53] *Linear versus nonlinear classifiers*, http://nlp.stanford.edu/IR-book/html/htmledition/linear-versus-nonlinear-classifiers-1.html, Accessed: 2016-04-24.

[54] *Kernel methods and nonlinear classification*, https://www.cs.utah.edu/~piyush/teaching/15-9-slides.pdf, Accessed: 2016-04-24.

[55] *A tour of machine learning algorithms*, http://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/, Accessed: 2016-03-11.

[56] *Ricky Ho predictive analytics: Neuralnet, bayesian, svm, knn*, http://horicky.blogspot.be/2012/06/predictive-analytics-neuralnet-bayesian.html, Accessed: 2016-04-18.

[57] *Scikit learn: 1.6. nearest neighbors*, http://scikit-learn.org/stable/modules/neighbors.html, Accessed: 2016-04-26.

[58] "Intelligent data engineering and automated learning - ideal 2007: 8th international conference, birmingham, uk, december 16-19, 2007. proceedings", in. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, ch. Using a Genetic Algorithm for Editing k-Nearest Neighbor Classifiers, pp. 1141–1150, ISBN: 978-3-540-77226-2. DOI: 10.1007/978-3-540-77226-2_114. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-77226-2_114.

[59] *K-means clustering*, http://www.onmyphd.com/?p=k-means.clustering&ckattempt=1, Accessed: 2016-04-26.

[60] *A tutorial on clustering algorithms*, http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/kmeans.html, Accessed: 2016-04-26.

[61] *Python pattern recognition: K-means*, http://pypr.sourceforge.net/kmeans.html, Accessed: 2016-04-26.

[62] *Scikit learn: One-class svm with non-linear kernel (rbf)*, http://scikit-learn.org/stable/auto_examples/svm/plot_oneclass.html, Accessed: 2016-04-26.

[63] A. Ng, *Stanford machine learning: Neurons and the brain*, https://class.coursera.org/ml-005/lecture/44, Accessed: 2016-04-26.

[64] ——, *Stanford machine learning: Model representation II*, https://class.coursera.org/ml-005/lecture/46, Accessed: 2016-04-26.

[65] *Stackoverflow what is a 'layer' in a neural network*, http://stackoverflow.com/questions/35345191/what-is-a-layer-in-a-neural-network, Accessed: 2016-04-18.

[66] ——, *Stanford machine learning: Multiclass classification*, https://class.coursera.org/ml-005/lecture/49, Accessed: 2016-04-26.

[67] *Openstar cnx neural network implementation*, https://cnx.org/contents/6CAkQax3@1/Neural-Networks, Accessed: 2016-04-18.

[68] ——, *Stanford machine learning: Model representation II*, https://class.coursera.org/ml-005/lecture/46, Accessed: 2016-04-26.

[69] ——, *Stanford machine learning: Cost function*, https://class.coursera.org/ml-005/lecture/50, Accessed: 2016-04-26.

[70] ——, *Stanford machine learning: Training a neural network*, https://class.coursera.org/ml-005/lecture/56, Accessed: 2016-04-25.

[71] ——, *Stanford machine learning: Autonomous driving*, https://class.coursera.org/ml-005/lecture/57, Accessed: 2016-04-26.

[72] *Quadcopter navigation in the forest using deep neural networks*, https://www.youtube.com/watch?v=umRdt3zGgpU, Accessed: 2016-04-26.

[73] A. C. Ian Goodfellow Yoshua Bengio, "Deep learning", Book in preparation for MIT Press, 2016, [Online]. Available: http://www.deeplearningbook.org.

[74] *Scikit learn: 1.10. decision trees*, http://scikit-learn.org/stable/modules/tree.html, Accessed: 2016-04-26.

[75] *Arjarapu blog space: Decision tree learning*, http://www.arjarapu.com/wordpress/2012/04/decision-tree-learning/, Accessed: 2016-04-26.

[76] *Scikit learn: 1.9. naive bayes*, http://scikit-learn.org/stable/modules/naive_bayes.html, Accessed: 2016-04-26.

[77] *Wolfram mathworld: Bayes' theorem*, http://mathworld.wolfram.com/BayesTheorem.html, Accessed: 2016-04-26.

[78] *Apriori algorithm*, https://en.wikipedia.org/wiki/Apriori_algorithm, Accessed: 2016-04-26.

[79] *R interface to oracle data mining*, http://www.analyticbridge.com/group/rprojectandotherfreesoftwaretools/forum/topics/r-interface-to-oracle-data, Accessed: 2016-04-18.

[80] A. Ng, *Stanford machine learning: Anomaly detection vs supervised learning*, https://class.coursera.org/ml-005/lecture/93, Accessed: 2016-04-26.

[81] *Normal distribution*, https://en.wikipedia.org/wiki/Normal_distribution, Accessed: 2016-04-26.

[82] *Online learning*, https://en.wikipedia.org/wiki/Online_machine_learning, Accessed: 2016-04-26.

[83] *Improving the identification performance of an industrial process using multiple neural networks*, http://article.sapub.org/10.5923.j.ajis.20120204.02.html, Accessed: 2016-04-26.

[84] *Cornell cs4670*, http://www.cs.cornell.edu/courses/cs4670/2015sp/lectures/lec34_datasets_web.pdf, Accessed: 2016-04-18.

[85] *Which "mean" to use and when?*, http://stats.stackexchange.com/questions/23117/which-mean-to-use-and-when, Accessed: 2016-04-22.

[86] ——, *Coursera machine learning, evaluating a hypothesis*, https://www.coursera.org/learn/machine-learning/lecture/yfbJY/evaluating-a-hypothesis, Accessed: 2016-04-22.

[87] *Model selection*, https://en.wikipedia.org/wiki/Model_selection, Accessed: 2016-04-22.

[88] *Overfitting and regularization*, hhttps://alliance.seas.upenn.edu/~cis520/dynamic/2014/wiki/index.php?n=Lectures.Overfitting, Accessed: 2016-05-12.

[89] *Advice for applying machine learning*, http://jay-mtl.github.io/Advice%20For%20Applying%20Machine%20Learning.html, Accessed: 2016-05-12.

[90] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection", in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, ser. SP '10, Washington, DC, USA: IEEE Computer Society, 2010, pp. 305–316, ISBN: 978-0-7695-4035-1. DOI: 10.1109/SP.2010.25. [Online]. Available: http://dx.doi.org/10.1109/SP.2010.25.

[91] A. Subaira and P Anitha, "Efficient classification mechanism for network intrusion detection system based on data mining techniques: A survey", in *Intelligent Systems and Control (ISCO), 2014 IEEE 8th International Conference*, IEEE, 2014, pp. 274–280.

[92] O. Al-Jarrah, A. Siddiqui, M Elsalamouny, P. D. Yoo, S. Muhaidat, and K. Kim, "Machine-learning-based feature selection techniques for large-scale network intrusion detection", in *Distributed Computing Systems Workshops (ICDCSW), 2014 IEEE 34th International Conference*, IEEE, 2014, pp. 177–181.

[93] *Netflow*, https://en.wikipedia.org/wiki/NetFlow, Accessed: 2016-04-28.

[94] *Ip api json format*, http://ip-api.com/docs/api:json, Accessed: 2016-04-28.

[95] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[96] S. García, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods", *Computers & security*, vol. 45, pp. 100–123, 2014. [Online]. Available: http://mcfp.weebly.com/the-ctu-13-dataset-a-labeled-dataset-with-botnet-normal-and-background-traffic.html.

[97] A. Sperotto, R. Sadre, D. F. van Vliet, and A. Pras, "A labeled data set for flow-based intrusion detection", in *Proceedings of the 9th IEEE International Workshop on IP Operations and Management, IPOM 2009, Venice, Italy*, ser. Lecture Notes in Computer Science, vol. 5843, Springer Verlag, 2009, pp. 39–50. [Online]. Available: https://traces.simpleweb.org/traces/netflow/netflow2/.

[98] A. Faes, *Machine-learning-techniques-for-flow-based-network-intrusion-detection-systems*, https://github.com/TheAxeC/Machine-learning-techniques-for-flow-based-network-intrusion-detection-systems, 2016.

# Appendix A

# User guide

## A.1  Running

The main program can be run using the command in Listing A.1. If no config file is passed as argument the program will look for a config file named "config.json" in the current directory.

```
python main.py [config-file-location]
```

## A.2  Running multiple tests

It is also possible to run multiple tests automatically. This can be done using the command as seen in Listing A.2. If no config file is passed as argument the program will look for a config file named "testing.json" in the current directory. The layout of the config file is fairly simple. It should be a list of config files that have to be run as seen in Listing A.2.

```
python testing.py [config-file-location]
```

```
1  [
2      "main/config.json"
3  ]
```

## A.3  Config file

The config file contains the main variables that define the execution of the program. If the format of an attribute is not correct, the program either skips that value or halts program execution.

There are five important boolean settings in the config file. **enable-IDS** is a boolean which specifies whether the IDS should run or not. **pcap-to-flow** is whether the small, built-in flow-converter should be used. This is used to convert *pcap* files to flow files. **print-labels** activates a section of the program which extracts and prints the labels that belong to a given dataset. This could be useful when trying to find out which labels the currently selected dataset will use.

The booleans **check** and **sniffer** can only be used when **enable-IDS** is set to *True*. **check** is used to activate or disable the checking of datasets after the training phase. **sniffer** is used to activate the built-in packet sniffer. The packet sniffer sniffs packets and builts flows, which are then fed to the machine learning algorithm.

```
1  {
2      "enable-IDS" : true,
3      "pcap-to-flow" : false,
4      "print-labels" : false
5
6      "check" : true,
7      "sniffer" : false,
```

**pcap-files** is a list of (source, destination) tuples for files that have to be converted using the built-in flow converter.

```
1      "pcap-files" : [
2          {
3              "src" : "../../test/dosattack.pcap",
4              "dest" :  "../../test/dosattack2.flow"
5          }
6      ],
```

**algorithm** is used to define which algorithm is used. The value of this attribute should match the name of one of the implemented machine learning algorithm classes. **featureClass** is used to select which class to use to retrieve features from a flow. **trainer** is used to select the training method. When the value is set to "Trainer", it is possible to select the training class in the data sets.

```
1      "algorithm" : "KNeighborsClassifier",
2      "featureClass" : "FlowFeature",
3      "trainer" : "Trainer",
```

Some attributes in the config file are used to save and use the models that the machine learning algorithms make. **model_dir** is used to define the directory location. **model** defines the name of the model file. **use_model** and **store_model** are respectively used to activate usage and saving of the model.

```
1      "model_dir" : "../../models/",
2      "model" : "model.mdl",
3      "use_model" : false,
4      "store_model" : true,
```

**data-sets** indicates the data to be used when training. The layout of each item in the list is dependent of which loader should be used. A netflow file has the attributes: **file**, **from** and **to**. An SQL trainer is also implemented. Here the attributes should indicate the **host**, the **user**, the **database** and the **password**. The **type** can be used to indicate which type of date it is. This should be the name of a trainer class.

The **check-sets** are used to check the performance of the machine learning algorithm. Here the **type** indicates which type the data is. The possible values are: "file" and "sql".

```
1      "data-sets" : [
2          {
3              "file" : "../../test/capture20110815.binetflow",
```

```
4                    "from" : 0,
5                    "to" : 40000
6             },
7             {
8                    "host" : "localhost",
9                    "type" : "SQLTrainer",
10                   "user" : "user",
11                   "db" : "dataset",
12                   "password" : "password"
13            }
14        ],
15
16
17    "check-sets" : [
18            {
19                   "file" : "../../test/capture20110815.binetflow",
20                   "type" : "file",
21                   "from" : 0,
22                   "to" : -1
23            }
24        ],
```

**feature-file** is a file that can contain information to be used in the feature class. The layout of this file is dependent on which "featureClass" is used. **logger** indicates the location of the log file to use. **good-labels** indicate which labels are considered to belong to normal behaviour.

```
1        "feature-file" : "features.json",
2        "logger" : "log.txt",
3        "good-labels" : "good.txt",
```

The following attributes are used in the small sniffer that was implemented. **protocol-file** is a file that selects which protocols to use. **timeout** is a value that define the amount of milliseconds to wait before closing a flow after the last packet in that flow.

```
1        "protocol-file" : "protocols.json",
2        "timeout" : 5000,
```

In order to be able to test multiple algorithms at a time, there is another method next to the *testing.py* file. During testing it was found that making a lot of config files was cumbersome work. A new feature was added to the config file. An attribute **configs** can be added. This should be a list.

Each element of this list, represents a distinct config file. The variables in the "outer"/"main" config file are considered the standard values. Each element in the list can overwrite these elements. For example, below the list contains two elements which have different algorithms. The data-sets and check-sets are the same for these two "config" files. **use_main** is used to let the program know whether to execute and "use" the outer config aswell, or to only use the configs list.

```
1        "use_main" : true,
2        "configs" : [
3            {
4                   "algorithm" : "KNeighborsClassifier"
5            },
6            {
```

```
 7              "algorithm" : "SupportVectorMachines"
 8         }
 9     ],
10  }
```

# Appendix B

# Developer Guide

The implementation is made to be modular. Adding new algorithms or features is as simple as adding a new class. Each module forms an important part of the implementation, such as the algorithm, the feature extraction and the data loaders.

## B.1    Machine learning module

New machine learning algorithms can be added to the implementation by creating a new class and placing the class in the *ml.py* file. A base class, **MLAlgorithm**, has to be inherited from and two methods have to be implemented:

```python
class SomeAlgorithm(MLAlgorithm):
    def train(self, data_set, target_set=None):
        data_set is an array of features
        target_set is an array of labels
            (or none for unsupervised learning)

    def predict(self, sample, corr=None):
        sample is a feature that is going
            to be used for prediction
        corr is the correct label that
            should be predicted
```

## B.2    Feature module

A feature class is used to parse a flow. The **Flow** class is a simple "struct" like class that contains the necessary member variables for a flow. The **Flows** class is a collection of flows. To make a new featureClass, a base class, **BasicFeature**, has to be inherited from and two methods have to be implemented:

```python
class FlowFeature(BasicFeature):
    def __init__(self, file):
        file is a json file that may contain extra
        information to make a featureClass
        more reusable

    def make_record(self, flow):
        flow is an instance of the Flow class
        return an array of numbers to be fed
            to a machine learning algorithm
```

## B.3   Loader module

The loader module is responsible for loading the flows from any external source. The exernal source could be a text file or a SQL database. The new class should be placed in the *loader.py* file. The loader class should be defined using a load and a get_netflow method. The parameters of load can be chosen at will since the loading has to be implemented in a training class.

```python
class LoaderName:
    def load(self, ...):
        load the data from the source

    def get_netflow(self):
        return the netflow
```

## B.4   Prediction Module

If a new type of loader needs to be used in the check-sets, a class has to be added to the Prediction module in the *prediction_type.py* file. A base class, **PredictionLoader**, has to be inherited from and a method has to be implemented: "load(self, data)". "data" is a dictionary loaded from the config file.

```python
class PredictionFile(PredictionLoader):
    def load(self, data):
        return the netflow
```

## B.5   Training module

The training module is responsible for all training classes. New types of training can be added to the implementation by creating a new class and placing the class in the *train.py* file. A base class, **Trainer**, has to be inherited from and a method has to be implemented. The method has several parameters. "algorithm" is an instance of a machine learning algorithm. "data" is a dictionary loaded from the config file from data-sets. "feature" is an instance of the selected featureClass. "good_labels" is a list of all labels that are considered normal behaviour.

*self.default* is a method implemented in the Trainer class. This method loads the file using the specified loader and file. Afterwards it extracts the features and feeds them to the machine learning algorithm.

```python
class TrainerName(Trainer):

    def train(self, algorithm, data, feature,
                                     good_labels):
        load the data
        return self.default(algorithm, loader,
                                     feature, file)
```

## B.6   Results module

The results module is the biggest module and is constructed from three files. The first file, *logger.py*, contains the main **Logger** class. These methods write to the log file that is specified in the config file.

*visualise.py* contains all methods that are used to visualise the machine learning algorithms. These visualisation methods can be injected into the program execution by using them in a machine learning class. This is required since the visualisation methods depend highly on the "train" and "predict" methods of the machine learning algorithms.

Finally, the *results.py* file receives all output from the prediction algorithms. They can filter out predictions that are considered normal behaviour and evaluate the machine learning algorithm using the amount of false positives and negatives.

## B.7 Other

*main.py* and *testing.py* are the main entry-points of the program. *main.py* puts together all different components to make the program execute. The *config.py* file is responsible for parsing the config file. Any new additions to the config file need to be reflected in this file. *predictor.py* is the file responsible for running the different check-sets. *sniffer.py* contains the small implementation of the sniffer, the real-time intrusion detection implementation.

# Appendix C

# Meetings

There are the reports for all meetings I have related towards my bachelor thesis. They are all in **Dutch**.

## C.1 Meeting 1: 09 Feb 2016

aanwezigen: Peter Quax, Bram Bonne, Pieter Robyns, Axel Faes

Dit is de eerste bijeenkomst met de begeleiders en promotor. Er is dus geen rapportering mogelijk van een vorige bijeenkomst. Tijdens de bijeenkomst is beslist om een *intruder detection system* te bestuderen en te implementeren.

De actiepunten die gedaan moeten worden:

- Beslissen voor wie het systeem gemaakt moet worden. Gaat dit voor end users zijn, of voor grote data centers. Hieraan hangt vast welke data (packets of netflow) gebruikt moet worden.

- Bekijken hoe machine learning algoritmes gebruikt kunnen worden in een *intruder detection system*.

- Bekijken wat netflow is.

- Er moet gekeken worden naar de manier waarop anomalies gegenereerd gaan worden om het systeem te testen/trainen.

Volgende afspraken zijn gemaakt:

- Er is gevraagd om te zorgen dat het systeem ook op correcte wijze informatie kan weergeven aan gebruikers. Tijdens het semester moet bekeken worden hoe deze weergave moet gebeuren.

- Libraries gebruiken indien mogelijk, om te vermijden dat het wiel opnieuw uitgevonden word.

- Er is de mogelijkheid geboden om aan de thesis te werken op het EDM.

- Er is afgesproken om *Overleaf* te gebruiken om de thesis in te schrijven.

- Een ruwe planning voor het werk moet gemaakt worden tegen 12 Feb.

- Een wekelijkse meeting is vastgelegd. Dit om 10:00 elke vrijdag.

- Begin mei moet een eerst draft van de thesis klaar zijn en eind mei moet de finale draft af zijn.

- Er moet een vulgariserende tekst gemaakt worden en een postersessie gegeven worden (op 29 juni).

## C.2 Meeting 2: 12 Feb 2016

aanwezigen: Bram Bonne, Pieter Robyns, Axel Faes

Dit is de tweede bijeenkomst met mijn begeleider. Netflow bevat op zichzelf niet zoveel informatie, maar het is toch handig om te kijken welke bevindingen gemaakt kunnen worden met deze data. Mogelijks kan er, indien gevonden wordt dat netflow alleen niet genoeg informatie bevat, ook gebruikt gemaakt worden van packet data.

Er is de mogelijkheid besproken om eventueel meerdere machine learning algoritmes te implementeren en te bekijken in welke situaties welke algoritmes beter werken.

De actiepunten die gedaan zijn:

- *Beslissen voor wie het systeem gemaakt moet worden.*: Dit gaat gedaan worden voor data centers

- Er zijn verschillende classificaties van machine learning algorithmes gevonden die gebruikt kunnen worden.

- Verschillende grote data sets van netflow en packets met sporen van anomalies zijn gevonden. Alsook programma's om verkeer te genereren.

Volgende actiepunten zijn besproken:

- Verder uitwerken van welke machine learning algoritmes gebruikt kunnen worden

- Bekijken netflow v9

## C.3 Meeting 3: 19 Feb 2016

aanwezigen: Bram Bonne, Pieter Robyns, Axel Faes

Professor Quax is aan het bekijken ofdat ik (gelabelde) netflow data kan verkrijgen van Cegeka. Dit zou heel handig zijn om mijn implementatie te testen op real world data.

Voorlopig moet ik enkel focussen op een passive intrusion detection systeem, geen preventie en niet direct inline in het netwerkverkeer. Ook de visualisatie moet later bekeken worden, de gebruiker is een netwerkadministrator. Er is tevens besproken dat python zelf mogelijks te traag is om packet sniffing op een goede snelheid uit te voeren. Hiervoor zou ik wireshark kunnen gebruiken (of de command line versie). Er is besproken om eventueel zelf datasets te genereren door malware te runnen op een VM of aparte machine.

De datastructuur voor de machine learning algoritmes is bekeken. Ik moet eens bekijken hoe de timestamps van de flowdata gebruikt kunnen worden. Om de effectiviteit (van de machine learning algoritmes) mogelijks te verhogen ga ik eens bekijken of ip-adressen ingedeeld kunnen worden in country-of-origin of iets dergelijks. Dit zou de machine learning algoritmes de mogelijkheid bieden om ook op deze parameter te bekijken of data malicious is of niet.

De actiepunten die gedaan zijn:

- Er is al een basis implementatie uitgewerkt voor het IDS

- De netflow structuur is bekeken en er is een datastructuur opgesteld die gefeed kan worden aan verschillende machine learning algoritmes.

- Progressie in de machine learning cursus: chapter 3 van de 18.

Volgende actiepunten zijn besproken:

- Beginnen aan de thesis: het schrijven van een hoofdstuk over machine learning en over hoe deze algoritmes toegepast kunnen worden op een intrusion detection systeem.

- Verder werken in de machine learning cursus.

- Ik moet eens bekijken ofdat ik een programma vind om pcap files om te zetten naar netflow. Anders moet ik dit zelf schrijven.

Ik heb ook een korte planning gemaakt van hoe de thesis eruit zou zien:

- Inleiding:

    - wat is een IDS
    - Waarom is er gekozen voor dit type IDS (host vs netwerk)
    - Waarom voor data centers
    - Waarom netflow
    - Waarom machine learning

- Wat is machine learning

- Hoe passen we machine learning toe op IDE en wat zijn de voor/nadelen

- Welke machine learning algortimes zijn wel/niet gebruikt

- Wat zijn de voor/nadelen van netflow

- Hoe met combinatie netflow/packets (Als dit gedaan zou worden)

- Welke data sets zijn gebruikt

- Wat zijn de bevindingen

- Hoe kan visualisatie/feedback gebeuren (richting admin en richting automatische preventie)

- Conclusie

## C.4   Meeting 4: 26 Feb 2016

aanwezigen: Bram Bonne, Axel Faes

Deze week is voornamelijk besteed aan de implementatie. Er is een netflow exporter geschreven. Er is bekeken ofdat timestamps gebruikt kunnen worden en ofdat ip-adressen opgedeeld kunnen worden per land. Er is besloten dat dit zeer weinig effect heeft op de accuraatheid van de machine learning algoritmes.

Momenteel zijn Support vector machines en K-nearest Neighbor Classifier algoritmes bekeken. Het K-nearest Neighbor Classifier algoritme is zeer efficient ( 98%).

In een later stadium kan bekeken worden om eventueel verdere analyse te doen op de data die malicious gevonden is, eventueel door pakketten te analyseren, of nogmaals door machine learning technieken. Er kan ook eens bekeken worden om een VM op te zetten, en daarin malware te runnen en dit verkeer te monitoren. Herbij zouden eigen datasets gegenereerd kunnen worden.

De machine learning cursus is gevolgd tot hoofdstuk 7. De cursus zou normaal af moeten zijn binnen 2 weken.

De actiepunten die gedaan zijn:

- Er is al een netflow exporter geschreven

- Er zijn experimenten uitgevoerd m.b.t de datastructuur die meegegeven wordt aan de machine learning cursus.

- Progressie in de machine learning cursus: chapter 7 van de 18.

- Er is begonnen aan de thesis.

- Het zou interessant zijn om eens te kijken ofdat ip-addressen opgedeeld kunnen worden in subnets.

Volgende actiepunten zijn besproken:

- Focussen op de thesis

- Verder werken in de machine learning cursus.

## C.5 Meeting 5: 04 Mar 2016

aanwezigen: Bram Bonne, Axel Faes

Deze week is voornamelijk besteed aan de thesis en aan het leren van de machine learning cursus. De machine learning cursus is gevolgd tot hoofstuk 10. De cursus zou tegen volgende meeting af moeten zijn. De algemene structuur van de thesistekst is nagekeken. Het hoofdstuk over "Attack classification" moet uitgebreid worden met een algemene uitleg over hoe aanvallen gedetecteerd kunnen worden. Het hoofdstuk over de gebruikte data-sets moet samengevoegd worden met het hoofdstuk dat de implementatie beschrijft. Het hoofdstuk over voor/nadelen van machine learning voor intrusion detection systemen moet verwerkt worden in het algemene hoofdstuk over machine learning.

De actiepunten die gedaan zijn:

- Verder werken aan ML cursus

- Schrijven aan thesis.

Volgende actiepunten zijn besproken:

- Verder werken aan ML cursus

- Schrijven aan thesis.

## C.6 Tussentijdse presentatie: 08 Mar 2016

aanwezigen: Maarten Wijnants, Peter Quax, Wim Lamotte, Jori Liesenborgs, Wouter vanmontfort, Pieter Robyns, Robin Marx, Bram Bonne, Axel Faes

Er is een tussentijdse presentatie geweest waarbij ik mijn huidige progressie moest tonen en een planning moest geven. De presentatie zelf is goed verlopen. Na de presentatie heb ik verschillende vragen gekregen.

Veel vragen die gesteld waren, waren bedoeld om te kijken of we het nut/doel van de bachelorthesis kennen en hoe we de invulling correct doen. Ook een belangrijk aspect is hoe het valideren van de correctheid van de experimenten die gedaan zijn/worden zal gebeuren.

Een opmerking was dat ik ook bestaande Intrusion detection systemen moet bekijken en ofdat deze machine learning gebruiken. Dan is ook belangrijk waarom ze het wel of niet gebruiken.

Er was verwacht dat ik al iets verder stond met de Machine learning cursus. Hierdoor kon ik niet altijd op de volledige diepgang de gestelde vragen beantwoorden. Ik begreep ook niet altijd de onderliggende vraag waardoor ik te oppervlakkig antwoorde. Qua machine learning algoritmes moest ik goed opletten voor overfitting en uitleggen hoe ik hiermee omga.

Er zijn ook vragen gesteld m.b.t mijn geplande extra om het intrusion detection systeem real-time te maken. Normaal wordt een flow pas doorgegeven als deze volledig afgesloten is, een mogelijke piste zou zijn om flows al te bekijken ook al zijn ze nog niet afgesloten. Ook het runnen van een VM met malware erop om zelf data-sets te generen is bevestigd dat een goed idee zou zijn. Als ik hiervoor infrastructuur nodig heb moet ik dit vragen.

Ik moet opletten met aanvallen die maar zeer weinig netwerktraffiek genereren. Ook moet ik goed beschrijven welke aanvallen wel of niet gedetecteert kunnen worden en uitleggen waarom. Dit staat momenteel al beschreven in mijn thesistekst. Ik zou ook mogelijkheden kunnen uitleggen die ervoor zouden kunnen zorgen dat ik toch alle (of een groot deel) van de aanvallen zou kunnen detecteren. Dit zou bv kunnen door toch packet-data te gaan bekijken.

Een algemene opmerking die gegeven was, was dat de presentatie visueler mocht zijn. Figuren en afbeeldingen zijn aangenamer om te tonen aan een publiek. Bij de postersessie moet er ook goed opgelet worden dat ik van persoon tot persoon bekijk hoe diep ik de materie uit mijn bachelorthesis kan uitleggen.

## C.7  Meeting 6: 11 Mar 2016

aanwezigen: Bram Bonne, Axel Faes

Deze week is voornamelijk besteed aan de thesis en het afmaken van de machine learning cursus. De machine learning cursus is volgens planning afgemaakt. De thesistekst moet ingestuurd worden zodat deze al nagekeken kan worden. Komende weken zal gespendeerd worden aan de implementatie en het uitzoeken hoe de flowdata gebruikt kan worden in de machine learning algoritmes.
De actiepunten die gedaan zijn:

- Afmaken Machine learning cursus

- Schrijven aan thesis.

Volgende actiepunten zijn besproken:

- Verdere implementatie algoritmes

- Bekijken hoe de flowdata gefeed kan worden aan de machine learning algoritmes

## C.8  Meeting 7: 18 Mar 2016

aanwezigen: Bram Bonne, Pieter Robyns, Axel Faes

Deze week is voornamelijk besteed aan implementatie. Er zijn verschillende algoritmes geïmplementeerd. Deze algoritmes zijn K-Nearest Neighbors, K-Means, Lineair Kernel Support Vector Machines met One vs All classification, Lineair Kernel Support Vector Machines met One vs One classification en Gaussian Kernel Support Vector Machines.

Er zijn verschillende experimenten uitgevoerd. De poort nummers zijn geïmplementeerd in 2 varianten. De eerste variant splitst de poorten in categoriën (veel gebruikte poort nummers en niet-veel gebruikte poort nummers) als een binaire feature. De andere variant maakt van elk poort nummer een nieuwe binaire feature (die stelt of de poort gebruikt is of niet). Deze variant geeft echter heel veel features, dit is enorm inefficient.

De gestelde feedback was om te kijken hoe goed het werkt als poort nummers als een continue feature voorgesteld wordt en om de categoriën op te spliten in >1024 en <1024. De IP-data can opgesplitst worden in aparte features voor IPv6, IPv4 en MAC. Deze data kan mogelijks voorgesteld worden als continue data. Er is voorgesteld om ook andere algoritmes te implementeren. Om gebruik te kunnen maken van datasets van Cegeka moet ik een presentatie maken en een afspraak maken met professor Quax.

Er is feedback gegeven op de thesistekst. Ik moet als eerste goed letten op spelfouten. De inleiding moet algemener uitgelegd worden. Ook niet gebruikte concepter zoals Signature-based IDS moet dieper uitgelegd worden. De attack classification maakt al teveel assumpties over wat er gedetecteerd kan worden. Dit zou eerst algemener uitgelegd moeten worden. Het machine learning hoofstuk bevat in het algemeen te weinig high-level beschrijvingen.

De actiepunten die gedaan zijn:

- Begin van hoe flowdata gebruikt kan worden

- Implementatie van K-Nearest Neighbors

- Implementatie van K-Means

- Implementatie van Lineair Kernel Support Vector Machines met One vs All classification

- Implementatie van Lineair Kernel Support Vector Machines met One vs One classification

- Implementatie van Gaussian Kernel Support Vector Machines.

Volgende actiepunten zijn besproken:

- Herschrijven en verwerken van feedback op de thesistekst

- Verdere implementatie: andere algoritmes

- Verdere implementatie: Ports indelen in >1024 en <1024

- Verdere implementatie: Ports indelen als continue data

- Verdere implementatie: IP indelen als continue data

- Verdere implementatie: Starttime instellen als unix time

- Maken presentatie voor Cegeka data set

## C.9 Meeting 8: 24 Mar 2016

aanwezigen: Bram Bonne, Pieter Robyns, Axel Faes

Deze week is voornamelijk besteed aan implementatie. Er zijn verschillende experimenten uitgevoerd. De poort nummers zijn geïmplementeerd zodanig dat er een indeling is tussen normale poorten (<1024) en speciale poorten (>1024). Er is ook gekeken ofdat poort data niet continu voorgesteld kan worden. Dit geeft echter slechtere resultaten dan te werken met de binaire indeling.

De IP-data kan opgesplitst worden in aparte features voor IPv6, IPv4 en MAC. Deze data is voorgesteld

als continue data. Het is moeilijk om IP-addressen zelf te gaan onderverdelen in categorieen of subnets. Het indelen als continue data geeft dan ook de beste accuraatheid. De starttijd is ook geïmplementeerd om te voegen als feature. Dit gebeurt door te bekijken in welke dag van de week/ uur van de dag en minuut van het uur de data gegenereerd word. Echter had deze feature geen goede invloed op de accuraatheid

Er zijn enkele algoritmes verder geïmplementeerd zoals een Decision Tree learner en een Naive Bayes algoritmes. Beide zijn supervised learning technieken. De Naive Bayes had nog goede accuraatheid, de Decision Tree Learner gaf slechtere resultaten.

De actiepunten die gedaan zijn:

- Verdere implementatie: andere algoritmes

- Verdere implementatie: Ports indelen in >1024 en <1024

- Verdere implementatie: Ports indelen als continue data

- Verdere implementatie: IP indelen als continue data

- Verdere implementatie: Starttime instellen als unix time

Volgende actiepunten zijn besproken:

- Herschrijven en verwerken van feedback op de thesistekst

- Testen van de implementatie

- Bekijken van Unsupervised learning algoritmes

## C.10   Meeting 9: 01 April 2016

aanwezigen: Bram Bonne, Peter Quax, Axel Faes

Deze week is voornamelijk besteed aan implementatie. Er is een nieuwe dataset gevonden. Deze dataset is afkomstig van de universiteit van Twente. Er was een honeypot opgesteld op het netwerk, alle data die hiermee gevangen is, is geclassificeerd en een dataset mee gemaakt. De dataset bevat voornamelijk externe aanvallen.

Er is ook gefocused op het trachten te gebruiken van unsupervised learning algoritmes. Echter heeft dit weinig opgebracht. Er kan wel op een accurate manier onderscheidt gemaakt worden tussen malicious en niet malicious data, maar het is moeilijk om vervolgens af te leiden over welk type malicious data het gaat.

De features die gebruikt worden in de machine learning algoritmes zijn nog eens overlopen. Professor Quax kwam met het idee om IP-adressen eventueel op te delen in origine (zoals land). Dit kan gebeuren via services zoals WhoIs.

De EDM dataet kan afgehaald worden bij het kantoor van professor Quax. Er moet ook zo snel mogelijk een meeting georganiseerd worden met Cegeka.

De actiepunten die gedaan zijn:

- Herschrijven en verwerken van feedback op de thesistekst

- Testen van de implementatie

- Bekijken van Unsupervised learning algoritmes

Volgende actiepunten zijn besproken:

- Verwerken data EDM

- Implementatie van WhoIs als feature

- Maken presentatie voor Cegeka data set

## C.11 Meeting Cegeka: 06 April 2016

aanwezigen: Peter Quax, Axel Faes, Cegeka

Er is een meeting geweest met Cegeka om de mogelijkheid te bespreken om data te kunnen gebruiken die afkomstig is van Cegeka, alsook om informatie te krijgen van de hudige IDS' die gebruikt worden.

In het begin is een korte presentatie gegeven waarin de eigenschappen van het te ontwikkelen systeem uitgelegd worden. Er wordt ook beschreven wat de algemene onderzoeksvragen zijn en hoe data nu gebruikt kan worden in machine learning algoritmes.

Een eerste vraag die gesteld is, is welke classificatie van 'onverwachte traffiek' er momenteel gebeurt in datacenter/hosting context. Cegeka werkt heel gelaagd. Voor de routers staat een DDoS protection systeem. Na de router staan zowel firewalls als SIEM devices. Voor grotere klanten is er extra beveiliging voorzien in de vorm van afgestelde IPS systemen en meer gedetailleerde threat detection. Zoveel mogelijk data wordt verwerkt, zowel flows als meer gedetailleerd. De systemen werken voornamelijk met een signature database en verwerken de data voornamelijk automatisch. De systemen moeten wel manueel afgesteld en onderhouden worden.

De classificatie gebeurt heel gedetailleerd door deze verschillende lagen. Er werd wel gesteld dat het veel interessanter is om outbound verkeer na te kijken in vergelijking met binnengaand verkeer. Zaken zoals port-scans zijn interessant om te weten maar gebeuren heel veel en kunnen al goed gedetecteerd worden.

Er was veel interesse naar een intrusion detectie systeem dat op basis van netflow en machine learning technieken werkt. Er accuraatheid van rond de 70-80 procent zou gezien worden als een goede accuraatheid. Ze hebben ook liever false positives dan false negatives. Teveel alerts genereren is heel vervelend, maar het is belangrijker dat voldoende anomalieen gedetecteerd worden.

Er is gevraagd of het mogelijk is om data te verkrijgen. Dit was zeker mogelijk. De netflow en corresponderende logs van 3 dagen wordt geleverd. De logs worden zowel in binair als text formaat geleverd. Het binair formaat kan ingelezen door een programma dat de logs visualiseerd. Dit programma wordt ook meegeleverd. Mogelijks zou ook output van het DDoS systeem geleverd kunnen worden. Om te bekijken ofdat een flow gezien is als malicious of niet moet dit bekeken worden of deze flow voorkomt in de logs.

## C.12 Meeting 10: 12 April 2016

aanwezigen: Bram Bonne, Axel Faes

Afgelopen week, woensdag 6 april, is de meeting geweest met Cegeka. Van deze meeting is een verslag gemaakt. Op maandag 4 april is de dataset van het EDM verkregen. Deze dataset bevat netflow data van het EDM netwerk van 18 februari tot 24 maart. Elke dag is netflow beschikbaar die voorgekomen is tussen 10u tot 24u. Deze zijn per 15 minuten gelogd in een file.

Er is kort gewerkt aan het verwerken van de data van het EDM. Dit gebeurt door de data te laten verwerken door verschillende algoritmes. De data waarvan vervolgens gesteld kan worden dat deze

daadwerkelijk malicious is, zal doorgegeven worden aan professor Quax. Op het moment zijn er nog niet genoeg testen uitgevoerd om iets te kunnen zeggen over de data.

Er is geïmplementeerd dat de country-of-origin van een IP ook gebruikt kan worden als feature voor de machine learning algoritmes. Er is gevonden dat dit voornamelijk werkt voor data die van buitenaf komt, zoals port scans. Er is ook kort al geprobeerd om visualisaties te maken van de machine learning algoritmes. Zulke visualisaties zouden interessant kunnen zijn om ook in de thesistekst te gebruiken. Tegen volgende bijeenkomst moet voornamelijk gewerkt worden aan de thesistekst. De actiepunten die gedaan zijn:

- Meeting Cegeka

- Implementatie van WhoIs als feature

- Implementatie van kleine visualisatie van algoritmes

- Verkrijgen data van EDM

Volgende actiepunten zijn besproken:

- Herschrijven en verwerken van feedback op de thesistekst

- Visualisaties van machine learning algoritmes zijn interessant voor thesistekst.

## C.13   Meeting 11: 18 April 2016

aanwezigen: Bram Bonne, Axel Faes

Ik heb mijn huidige vooruitgang laten zien van de bachelorproef tekst. Hierbij is een korte herschikking gekomen van de hoofdstukken. Ik moet het "Implementatie" hoofdstuk opsplitsen naar een hoofdstuk "Implementatie" en een hoofdstuk "Evaluatie". In "Implementatie" moet mijn implementatie zelf beschreven staan, in "Evaluatie" moeten de resultaten beschreven worden.

Verder is kort uitgelegd wat de algemene structuur is van het machine learning hoofdstuk en welke aanpassingen er gebeurd zijn. Het hoofdstuk is opgesplitst in meerdere delen zodanig dat de uitleg, de algoritmes en validatie in aparte hoofdstukken uitgelegd wordt. Het hoofdstuk "Preventie" moet ik aanpassen naar iets gelijkaardigs aan "Future work". Het hoofdstuk "Visualisatie" moet samengevoegd worden met "Implementatie". De actiepunten die gedaan zijn:

- Verwerken feedback op machine learning hoofdstukken.

Volgende actiepunten zijn besproken:

- Herschrijven en verwerken van feedback op de thesistekst

- Halen data bij Cegeka

## C.14   Meeting 12: 22 April 2016

aanwezigen: Bram Bonne, Axel Faes

Ik heb mijn huidige vooruitgang laten zien van de bachelorproef tekst. De meeste tekst is al stukken beter. Er mogen bij enkele stukken nog numerieke voorbeelden komen te staan. Deze stukken zijn de regularisatie en de feature scaling. Het hoofdstuk over "Algorithms" moet nog herschreven worden. Tevens ga ik een "Summary" schrijven op het einde van het machine learning hoofdstuk zodanig dat nog snel en kort een samenvatting gegeven wordt.

De actiepunten die gedaan zijn:

- Verwerken feedback op machine learning hoofdstukken.

Volgende actiepunten zijn besproken:

- Verwerken feedback en herschrijven "Algorithms" tegen zondag 24 April

- Schrijven van tekst

## C.15   Meeting 13: 29 April 2016

aanwezigen: Bram Bonne, Axel Faes

Het herschrijven van het "Algorithms" hoofdstuk heeft langer geduurd dan gedacht en is pas ingeleverd op woensdag 27 april 2016. Tijdens de meeting is toegelicht welke aanpassingen gebeurt zijn aan de thesis. Er is kort besproken dat er ook focus gelegd moet worden over het verwerken van de data van Cegeka. Dit zou al belangrijk zijn voor de eerste draft van de thesis.

De actiepunten die gedaan zijn:

- Beginnen met verwerken van data van Cegeka

- Schrijven korte summary op het einde van het machine learning hoofdstuk

- Numerieke voorbeelden plaatsen bij regularisatie en feature scaling

- Hoofdstuk "Algorithms" herwerken

Volgende actiepunten zijn besproken:

- Schrijven hoofdstuk IP Flows

- Herwerken van hoofdstuk "Machine learning for an IDS"

- Verder verwerken dataset Cegeka

## C.16   Meeting 14: 04 Mei 2016

aanwezigen: Bram Bonne, Axel Faes

Er is feedback gegeven over de machine learning hoofdstukken. De inleiding is goed geschreven. Bij het gedeelte van Feature scaling moet ik 2 voorbeelden omwisselen. Bij de cost function voor logistic regression zou nog een kort voorbeeldje bij moeten. De uitleg over kernels bij Support Vector Machines moet duidelijker verwoord worden. Het gedeelte over distances kan verplaatst worden naar een ander hoofdstuk. Bij Clustering zou pseudocode toegevoegd mogen worden. Decision tree algorithms en Bayesian algorithms moet beter uitgelegd worden. Verder zijn er nog enkele kleinere opmerkingen. Er wordt geprobeerd om al deze feedback te verwerken voor de deadline van de eerste draft (16 mei 2016)

De actiepunten die gedaan zijn:

- Schrijven hoofdstuk IP Flows

- Afmaken Attack Classification hoofdstuk

- Herwerken van hoofdstuk "Machine learning for an IDS"

- Verder verwerken dataset Cegeka

Volgende actiepunten zijn besproken:

- Feedback verwerken van machine learning hoofdstukken

- Afmaken "implementatie" hoofdstuk

- Afmaken "evaluatie" hoofdstuk

- Afmaken "Intrusion detection systems" hoofdstuk

- Schrijven inleiding hoofdstuk

- Schrijven conclusie hoofdstuk

- Schrijven Future work hoofdstuk

- Schrijven van Nederlandstalige samenvatting

- Verder verwerken dataset Cegeka

# Appendix D

# Review

In the past semester I have learned a lot. Together with professor Peter Quax, professor Wim Lamotte and Bram Bonne, I came up with the topic for this thesis. Personally I wanted to combine machine learning and cyber security. I did not have a deep understanding of either topic. However, I still pushed on and picked this topic.

I started following a machine learning course from Coursera and started reading up on intrusion detection systems. I quickly realised that there was a lot of work to do. My weekly meetings with Bram Bonne allowed me to ask any questions I might have and stay on schedule with my work.

Eventually I got my first prototype of the implementation running and I could start testing. This was quite difficult since at that time I still didn't properly understand machine learning algorithms. Once I finished the machine learning course, testing became a lot simpler to do since I knew how each algorithm worked and how it should be used.

In the end, I have broadened my knowledge of both machine learning and intrusion detection and I had the chance to test my implementation with data received from Cegeka. It was a lot of work, as told by my mentor and advisors, but I am glad that I chose this topic.