CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Real-time anomaly detection in computer networks using machine learning

An implementation of unsupervised machine learning algorithms on real-time network traffic

Master's thesis in Computer Science and Engineering

ALEXANDER BRANZELL

PER NILSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2019

Master's thesis 2019

Real-time anomaly detection in computer networks using machine learning

An implementation of unsupervised machine learning algorithms on real-time network traffic

ALEXANDER BRANZELL
PER NILSSON

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2019

Real-time anomaly detection in computer networks using machine learning
An implementation of unsupervised machine learning algorithms on real-time network traffic
ALEXANDER BRANZELL
PER NILSSON

Supervisor: Erland Jonsson, Department of Computer Science and Engineering
Advisors: Fredrik Larsson, Jonas Berg, SAAB AB
Examiner: Magnus Almgren, Department of Computer Science and Engineering

Master's Thesis 2019
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The all seeing eye, a famous symbol of surveillance rendered in ASCII text

Typeset in LaTeX
Gothenburg, Sweden 2019

Real-time anomaly detection in computer networks using machine learning
An implementation of unsupervised machine learning algorithms on real-time network traffic
ALEXANDER BRANZELL
PER NILSSON
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This thesis explains how to employ machine learning methods for anomaly detection in real-time on a computer network. While using machine learning for this task is not a novel concept, little literature is on the subject of doing it in real time. Most machine learning research in computer network anomaly detection is based on the KDD '99 data set and aims to prove the efficiency of the algorithms presented. The focus on this data set has caused a shortage of scientific papers explaining how to gather network data, extract features and train algorithms for use in real time networks. It has been argued that using the KDD '99 data set for anomaly discovery is not applicable to real time networks. This thesis proposes how the data gathering process can be done using a dummy network and compares the results of k-means clustering, one class SVM and LSTM neural networks with reported results of the same algorithms on the KDD '99 data set. The results show that algorithms trained using the KDD data set have worse accuracy, but that this can be linked to the lack of complexity in the gathered data.

Keywords: Network Security, Anomaly Detection, Real-Time, Computer Networks, Machine Learning, Time Series, Data Generation.

Acknowledgements

We would first like to say thank you to our supervisor at Chalmers Univeristy of Technology, Erland Jonsson for helping us complete this master thesis. We would also like to say thanks to our examiner Magnus Almgren for valuable input when writing this master thesis.

We are also grateful for our supervisors at SAAB AB, Fredrik Larsson and Jonas Berg for helping us with the lab setup, providing us with the required equipment and giving us feedback during the project.

Alexander Branzell, Per Nilsson, Gothenburg, June 2019

Contents

List of Figures

List of Tables

1
Introduction

Computer security is an ever evolving issue as technology becomes increasingly powerful. The techniques for protecting information and information systems of yesterday can not assure the security of today's systems. A vital part of computer security is knowing if a system has been or is being tampered with. These systems are called intrusion detection systems and are created to mitigate the risks of system failure and misuse [44]. Intrusion detection systems can be divided into anomaly based systems and signature based systems. Signature based systems work by manually creating signatures that check for known attacks or malfunctions. The problem with this type of system is that new or unknown attacks are impossible to handle. Recent advances in network technology and the public availability of sophisticated hacking tools allow users with little to no technical experience to commit complex network attacks [17] [21]. These attacks have become hard to identify and creating signature based detection is a cumbersome and expensive process. An anomaly based detection system could be a lot more effective in this case [37]. Anomaly based systems instead try to identify what a normal state of the system looks like and reports when the system is not in this state. These systems are effective at identifying unknown system behaviour that can be from either an attack or a system failure. Anomaly based systems are often used in conjuncture with signature based security systems to safeguard against as many threats as possible.

Machine learning has been used for anomaly based intrusion detection systems for decades. These types of algorithms are good at identifying patterns in data, which makes them perform well as a part of an anomaly based intrusion detection system. However, a lot of research on the algorithms in this field relies on using a single data set, namely the KDD '99 data set [39] [37]. This data set is now 20 years old and

in the last two decades computer networks and attacks have evolved immensely. The KDD '99 data set is based on the 1998 DARPA intrusion detection evaluation program's collected data [1]. The data set was collected over 9 weeks on a network created to emulate typical U.S air force LAN network traffic. An overview of the KDD '99 data set features can be found in Appendix A.1. The data set does not contain data based on individual packets transmitted over the network. Instead each packet is assigned to a connection from where the data is derived. A connection is defined as a sequence of TCP packets starting and ending at some well defined time. This data set does not contain any other protocol than TCP. Even though the KDD '99 data set is widely used it has been criticized for having a lack of quality [4] [16] [41]. The quality of the data set is essential in machine learning and it has been argued that this devalues papers based on the KDD '99 data set. While having a data set as diverse as the KDD '99 data set is good, there exists a gap in academia showing the process from collecting network traffic to implementing a live system. This project aims to help fill this gap by looking at processing said traffic, training algorithms by using the traffic as data and finally evaluating the algorithms on a real-time network documenting how this can be accomplished.

This thesis is divided into seven chapters. Chapter 1 provides information on the problem and describes the layout of the thesis. Chapter 2 provides an explanation on key concepts as well as the techniques used in the project. Chapter 3 explains what specific methods together with the techniques shown in the background Chapter were considered and why some were chosen over others. Chapter 4 shows how the techniques explained Chapter 3 were deployed to create the anomaly detection system. Chapter 5 displays the outcome of the project. Chapter 6 contains the authors' reflections on the algorithms implemented and the ethics of deploying a surveillance system on a computer network. Chapter 7 contains ideas for future projects and the authors' final thoughts on the project.

1.1 Aim

This section explains the intent of writing this master thesis. As stated earlier in this report, using the KDD '99 data set is not a viable option in modern research on using machine learning for intrusion detection. These types of data sets are efficient for bench-marking the accuracy of algorithms [16] but models trained using this data has no application on a real live network [13]. There is an inherent flaw in using a pre-generated data set as the data will be tailored to the origin network and network traffic varies between computer networks. By rejecting the KDD '99 data set this thesis aims to implement machine learning algorithms on a real-time network and to propose a procedure of doing this. The process consists of generating, collecting and pruning data, training and implementing the algorithms and developing the anomaly detection system. The data gathering, processing and evaluation is on synthetic network traffic data generated in the lab setup. The real-time anomaly detection algorithm is evaluated on the same lab setup.

1.2 Goal

In this section the preferred results of the project are presented. By using the same algorithms found in reports on anomaly detection using the KDD '99 data set, the goal is to achieve similar or higher accuracy when applied live on the lab network. To measure accuracy the metrics of F1-score, precision and recall value are used. These metrics are explained further in Section 2.7. The following list shows how this main goal was divided into sub goals and what section explains how these sub goals were completed.

- Create a lab for data and anomaly generation, Sections 2.3, 3.2, 4.2
- Find and develop pre-processing techniques for the gathered network data, Sections 2.4, 2.5, 3.4, 4.3
- Find and develop machine learning techniques to detect anomalous data, Sections 2.6, 3.5, 4.4

By completing these sub goals the main goal of the project is completed.

1.3 Limitations

This section goes through the restrictions and limitation for the master thesis to give the reader a clear scope of what the focus of the project is and what is investigated. The project does not focus on exploring different attack vectors against a network with anomaly detection. Different ways of creating anomalies will be implemented but there is no deeper investigation of the attacks themselves. To make the generation, gathering and pre-processing of network traffic data easier, the network handles only a subset of all network traffic. Since a key aspect of this project is to not use KDD '99 it should be noted that no other generated data set is considered either.

1.4 Related work

This section shows some of the papers that are similar to this project. Some of this research perform machine learning techniques on the KDD '99 data set. This research will be used as a metric for evaluating the results of using the same machine learning techniques on live network traffic. There are papers in this section showing how to use collected network traffic for machine learning. Both traffic collected by the authors of the papers and from the DARPA 1998 data set has been used. Most of these projects are never tested on a live network. The projects that are tested on a live network use different algorithms and pre-processing techniques than the ones presented in this thesis.

1.4.1 Data generation

In order to perform machine learning on network traffic useful data must be extracted from the network connections. Key features of the connections made can be extracted from the IP header of a network packet. Benferhat et al. describe a set of features from the DARPA data set that are useful [9]. The report describes how to translate network traffic into a data set of connection records. The intention is to map the network traffic into a data set useful for intrusion detection. There is no mention of whether these features would be useful in a machine learning context or not. The connection records created are similar to those in the KDD '99 data set, but use a different set of

features. Cao et al. describe a process of creating a data set similar to the KDD '99 data set with the intention of being used for machine learning and knowledge discovery[13]. The article does not bring up how the data was generated or what tools were used for anomaly generation. The network traffic was pre-processed into connections like in the KDD '99 data set. The features presented in these two papers together with the features found in the KDD '99 data set give an accurate idea on what features are necessary for anomaly detection in computer networks. The collected traffic needs o be pre-processed into data that is useful for machine learning. Madan et al. [27] translate network traffic into time series which the authors then use to predict network traffic. This type of pre-processing should allow a machine learning algorithm to predict whether the time series is anomalous or not.

1.4.2 Machine learning on KDD '99

Several machine learning algorithms have been employed on the KDD '99 data set. Not all of these algorithms are fit for anomaly detection and not all are fit for deployment in a live network. Zhang et al. propose the use of a One-class support vector machine for anomaly detection [45]. The authors evaluate their algorithm on the KDD '99 data set and compare the results to a probabilistic neural network and a regular support vector machine. The paper shows that the One-class SVM is apt for anomaly detection in computer networks. The authors mention in their conclusion that improved data would increase the algorithms performance and that this is something that future research should investigate. Judging by the performance of the one class support vector machine in this paper it should perform well when evaluated on a live network.

Jianliang et al. suggest a method of using the k-means clustering algorithm to find the centers of the normal data clusters in the KDD '99 data set [22]. By using the Euclidean distance between these cluster centers and different data point it is possible to find if they are anomalous or not. Han transforms the KDD '99 data set to a data set based on information entropy in order to use the k-means Algorithm [19]. Information entropy is used to identify initial cluster centers for the

k-means algorithm. The authors compare their algorithm to a regular k-means algorithm and a modified dynamic k-means algorithm and find that the information entropy surpasses both. The simplicity of the k-means clustering algorithm should allow it to perform well in a live network.

Thi et al. use a Long Short Term Memory Recurrent Neural Network (LSTM or LSTM-RNN) and time series data based on the KDD '99 data set to find anomalies in the data set [42]. By predicting the next second of the time series and then measuring the error rate the authors are able to classify whether the time series data points are anomalous or not. The results are only attained using the KDD '99 data set and it is never tested on a live network.

1.4.3 Machine learning on captured data

As previously mentioned most research on machine learning for anomaly detection in computer networks is done using the KDD '99 data set. However, Kumar et al. show a process where the authors gather network data to use in machine learning [38]. They use a Naïve Bayes classifier to predict whether the gathered traffic was anomalous or not. The classifier is only tested on a gathered data set and never implemented in a live network. The authors mention that future research should attempt different machine learning algorithms and perform testing on a live network. Cao et al. show a process where they efficiently employ machine learning on a live network [13]. The results were measured against results gathered from the KDD '99 data set. The machine learning algorithms that were tested were an artificial neural network and fuzzy c-means. The authors mention that future research should attempt using different machine learning algorithms.

2

Background

This chapter explains the key concepts that are used in this project. The purpose is to give the reader an overview of the different tools, techniques and theories needed to complete the project. Section 2.1 gives a brief explanation of network intrusion detection systems. Section 2.2 contains information on the network anomalies that are used in this project. Section 2.3 explains the generation of synthetic network traffic data with different tools. Section 2.4 explains what features are interesting. Section 2.5 explains how data can be handled to facilitate the machine learning algorithms. Finally, section 2.6 explains the three algorithms LSTM, SVM and k-means clustering and the theory behind them.

2.1 Network Intrusion Detection Systems

Network Intrusion Detection Systems, NIDS, are used to detect malicious traffic in a computer network. The subject can be divided into two types, signature based detection and anomaly based detection [44] [15]. Signature based NIDS follow a static set of signatures and patterns to detect already known attacks. Anomaly based NIDS are used to classify normal traffic and abnormal traffic, where the latter is considered to be an anomaly in the network. They do this by using a set of rules and heuristics and these are the types of NIDS this project focuses on. Typically anomaly based NIDS are developed by gathering data of the system and feeding it to an algorithm with the intention that the algorithm should be able to distinguish normal and anomalous data [33].

2.2 Network Anomalies

Anomalies are per definition unknown entities. In computer networks an anomaly would be unexpected network traffic or traffic flows [7]. Anomalies can appear in any network for various reasons and do not necessarily mean that an intruder or malicious user has tampered with the network. Examples of anomalies could be a sudden unexpected spike in network congestion due to a router failure or a denial of service attack sending many more packets to a node than normal. Below are examples of three different network anomalies,

- Node Failure
- Denial of Service attack
- Port Scan

which are all explained in this section.

2.2.1 Node Failure

Node failure is when a node in the network suddenly stops responding. This can be because of a power outage, a hardware or software failure in the node or because of any other reason. The node is simply not active in the network any longer. This can only be discovered when using TCP traffic, but is undiscoverable when using UDP due to its stateless nature. This type of anomaly is easy to reproduce by simply disconnecting the node from the network. This is the only anomaly of the three in this project that is not included in the KDD '99 data set. It is included in this project to diversify the anomalies to not only contain attacks but to also include other anomalous network behaviour.

2.2.2 Denial of Service (DoS) attack

Denial of service attacks work by targeting the victims ability to respond to communication requests [34]. There are several ways to carry out these types of attacks but a basic DoS attack is to send a large amount of TCP SYN packets to a known service on the host. The victim believes that these packets are made to initiate a TCP handshake, when the victim responds the attacker never follows up on the handshake, causing the victim to spend resources waiting for more communication. The idea is to exhaust the victim's resources such

that the victim is no longer able to respond to legitimate communication requests. DoS attacks are no longer very effective as computers have become more powerful. However, the distributed DoS attack is still viable and is a common network attack [10]. The distributed DoS attack is a synchronized DoS attack running on multiple attacking machines, making the amount of traffic much larger. These types of attacks can also be used to increase network congestion and cause major slowdowns in network traffic. A DoS attack causes a noticeable difference in network traffic, as there is a sudden increase in connections to a specific port that are never resolved.

2.2.3 Port Scanning

Port scanning is a technique where the user is attempting to find out what services are running on a remote machine. This is usually done to find vulnerable ports that the targeted machine is listening to and either close them, add security layers on top of the services or, in the case of a malicious user, exploit these ports. A port scan is performed by sending a packet to the port in question and infer information depending on how the target responds. This is usually noticeable when looking at the network traffic, as one host will send out many packets to a lot of different ports to see how the targeted machine reacts. Port scans are common before launching an attack over a network [12].

2.3 Network traffic generation

Network traffic varies depending on what services are running on the nodes in the network and the network topology. Preferably the network communications should be made to imitate some type of predefined system. This traffic can be created by using tools to create synthetic data, replaying pre-recorded network data or by having people perform actions over the network simulating real users. Synthetic data is data that has the same features and behaves the same way as real data but does not actually contain any significant information. The information it carries is usually randomized to give the impression that it is real. Recorded data is data that is collected earlier and then re-played over the network. It causes the computer to replay a

saved communication as if it was happening again. Human data is having human actors using the communicating machines to send different types of data and handling them differently. This could, for example, be a person browsing a web server or two people using a peer-to-peer chat client. Below is a short description of three network traffic generation tools.

2.3.1 Distributed Internet Traffic Generator (D-ITG)

D-ITG is a platform used to generate realistic network traffic. This tool is capable of using a variety of stochastic processes to simulate real time network performance with varying delays and packet sizes using a large variety of probability distributions [8]. D-ITG can imitate a large number of different applications over the network, such as online games or voice communication. It uses client server functionality to send and receive network communication and also keeps a log over all network communication that has been sent and received by D-ITG. More information on D-ITG can be found on their website [2].

2.3.2 Iperf

Iperf, also known as iperf3 is an open source tool for bandwidth benchmarking in computer networks. It uses client server functionality to setup connections and is able to send packets using a variety of different protocols and parameters. When connecting to a server using iperf, it will attempt to send as much data as possible to the server and report back its throughput. More information about iperf can be found on the iperf website [3].

2.3.3 SourcesOnOff

SourcesOnOff was developed to help network engineers generate network traffic for testing and evaluation of different applications on computer networks. It attempts to generate the same type of traffic a network administrator would see in a local area networks and on the internet using client server functionality [43]. It supports sending packets with different delays and sizes using various probability distributions to mimic realistic network performance.

2.4 Feature Selection

Data must be ordered and labeled clearly to be considered useful when being collected and processed. This means that all data gathered must be sorted and put into context. When talking about feature selection the intent is to extract a number of characteristic features that would describe the data accurately. This allows the algorithms to process the data in a way that would make it useful for machine learning. There are several ways this can be done and it depends on how the data processing algorithms work. There are however indicators that help when selecting features such as variance. For example, a feature with zero variance would not be useful since it would never change and would thus be redundant.

2.5 Pre-processing

In order for an algorithm to work correctly it requires a numeric representation with high quality data set i.e. a data set that the algorithms understand with minimal inaccuracies [6]. The data can contain irregularities or other skewed information that would cause the algorithm to perform poorly. The pre-processing step is usually done by filtering out faulty data points. This can be done by removing the faulty data or correcting it. There is more to pre-processing data than filtering, such as normalizing the features since they can vary between a large range of values.

2.6 A brief introduction to three machine learning algorithms

Machine learning uses gathered data to train an algorithm to recognize patterns. Training is done by feeding recurring data patterns to the algorithm in order to increase its understanding of the data's features, allowing it to process new data accurately. This can be done using both supervised and unsupervised learning. The unsupervised learning will categorize unknown, unlabeled data whereas the supervised requires categorized and labeled data. In anomaly detection supervised learning can cause problems since unknown anomalies are not in

a labeled data set and thus the trained network will not be able to detect these new attacks. However, if labeled data is available supervised methods are good at finding correlation between the different features.

The theory behind three different machine learning algorithms are introduced in this section. They are:

- Feed-forward Neural Network and Long Short Term Memory
- Support Vector Machine
- k-means Clustering

The Long Short Term Memory Neural Network is a more complex structure of the Feed-Forward Neural Network which is first explained in the subsection below.

2.6.1 Feed-Forward Neural Network

An example of supervised machine learning is the feed-forward neural network. It is created using layers of nodes which are often referred to as neurons [28], where each neuron is connected by weights to the neurons in the previous and next layer. By adjusting these weights the neurons are able to detect features in the input data. The more layers in the neural network the more complex features can be detected. Applying networks like these with many intermediate layers is referred to as deep learning.

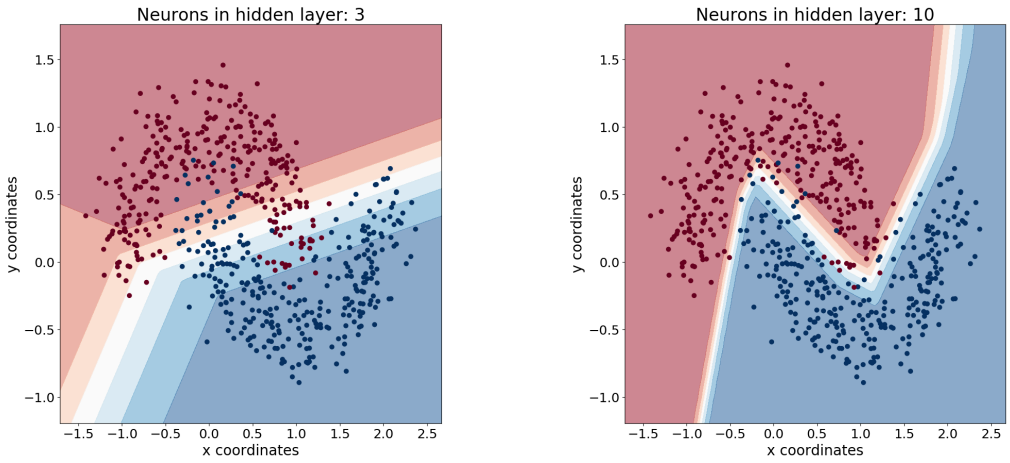
The structure of a neural network is made up by artificial neurons. The neurons are placed in three different kinds of layers, the input, the hidden and the output layer. The input layer simply consist of an m-dimensional input vector, $\vec{\xi}$, fed to the network. The input values are propagated to the next layer, called the first hidden layer through weights $\mathbf{W}$. Looking at Equation (2.1), each neuron, j, in the hidden layer calculates a weighted sum of the previous layer together with a bias, θ_j. An activation function σ, such as *tanh*- or the *sign*-function, is applied on the product to force the output to be between two well defined values and create an output, ζ_j.

$$\zeta_j = \sigma\left(\sum_{i=0}^{m} \xi_i W_{ji} + \theta_j\right) \tag{2.1}$$

A neural network can consist of zero to an arbitrary number of hidden layers. By adding more hidden layers or number of neurons, the

network might be able to classify more complex patterns of data as can be seen in Figure 2.1.

The output layer is the last layer in the neural network. As in the hidden layers the neurons in the output layer computes a weighted sum from previous layers, add a bias and use an activation function on the product. The output layer serves as the layer that is supposed to classify the input data to a final output o.



(a) Three neurons in the hidden layer. **(b)** Ten neurons in the hidden layer.

Figure 2.1: The figures show two different neural networks and how increasing the number of neurons in the hidden layer can separate and classify more complex data.

Backpropagation

By using labeled data the weights and biases in the network are adjusted using backpropagation [31]. To be able to back propagate the classification errors, one needs to define a loss or energy function of the output. By defining the loss function as the mean squared error one gets a convex function which can be minimized.

$$H = \frac{1}{2n} \sum_{i=1}^{n} (o_i - \iota_i)^2 \tag{2.2}$$

In Equation (2.2) the energy value H is the mean squared error of the predictions of the network o_i and the actual class ι_i. The predicted output o of the network for an input ξ is a sum of values from all

previous layers. For a neural network, with one hidden layer with k neurons and m-dimensional input vector, the output is calculated by the sum of previous layers and a bias (2.3). Each node i in the hidden layer is in its turn a weighted sum of the input layer (2.4). Combining these will give the full equation for the output layer in equation (2.5).

$$o = \sigma\left(\sum_{i=1}^{k} W_{1i}^{(2)}\zeta_i + \theta_1^{(2)}\right), \text{ where} \tag{2.3}$$

$$\zeta_i = \sigma\left(\sum_{j=1}^{m} W_{ij}^{(1)}\xi_j + \theta_i^{(1)}\right), \text{ such that} \tag{2.4}$$

$$o = \sigma\left(\sum_{i=1}^{k} W_{1i}^{(2)}\sigma\left(\sum_{j=1}^{m} W_{ij}^{(1)}\xi_j + \theta_i^{(1)}\right) + \theta_1^{(2)}\right) \tag{2.5}$$

By using the gradient descent method, $\boldsymbol{x}(t+1) = \boldsymbol{x}(t) - \eta\nabla H\left(\boldsymbol{x}(t)\right)$, where H is the loss function (2.2) and $\boldsymbol{x}$ is a unit that is updated with respect to the gradient of the loss function. η is the step length where a large step length could result in not finding the global minima, while a small step length can get stuck in a local minima. This process is repeated until it converges. To update a specific weight in an arbitrary layer, $W_{lm}^{(k)}$, one takes the partial derivative of the loss function with respect to the specific weight.

$$\frac{\partial H}{\partial W_{lm}^{(k)}} = \frac{1}{2n}\frac{\partial \sum_{i=1}^{n}\left(o_i - \iota_i\right)^2}{\partial W_{lm}^{(k)}} \tag{2.6}$$

A gradient descent step then becomes:

$$W_{lm}^{(k)}(t+1) = W_{lm}^{(k)}(t) - \eta\frac{1}{2n}\frac{\partial \sum_{i=1}^{n}\left(o_i - \iota_i\right)^2}{\partial W_{lm}^{(k)}} \tag{2.7}$$

The number of inputs, i, fed to the neural network can be a single one, a batch or the entire data set. To prevent the neural network to overfit, i.e. training too much on a limited set of data points such that the algorithm no longer generalizes well, the batch or sample can be randomly chosen for each iteration. One can also split the dataset into a training part and a validation set. The validation set is meant to be an unbiased validation of the training process. By monitoring its accuracy it is possible to see when the network overfits on the training data.

The Long Short Term Memory Neural Network (LSTM)

A disadvantage in many machine learning algorithms is that they look at each data point separately. When looking at a sequence of data the previous data points can hold a great deal of information which can be used to predict the current point. LSTM is an extension of recurrent neural networks, RNN. In a basic RNN, each neuron uses its value from the previous point in time to calculate the current output as can be seen in Figure 2.2.

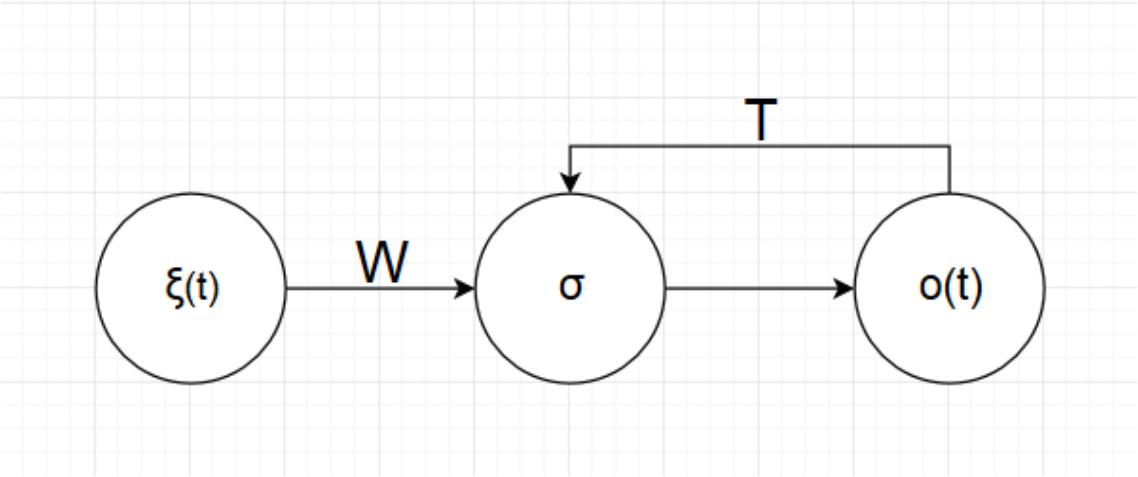


Figure 2.2: Graphical interpretation of a basic recurrent neural network, RNN. The middle node is the neuron which applies an activation function, σ, on a weighted sum of the input $\xi(t) = \xi_t$ and the feedback from previous iterations $o(t-1) = o_{t-1}$ with the weight matrices W and T.

Let o_t be the output of a neuron that saves its value from previous iteration. In the next iteration it will use the normal input ξ_t multiplied with a weight matrix W, as in a feed forward neural network. It will also add the previous value o_{t-1} multiplied with a transition matrix, T, as can be seen in Equation (2.8).

$$o_t = \sigma\left(W\xi_t + To_{t-1}\right) \tag{2.8}$$

An arbitrary activation function, σ, is then applied to this sum. By training a RNN one objective is to tune the transition matrix such that the network learns if previous points were of interest or not.

The RNN has a problem with vanishing gradient, meaning the network will have problems in remembering longer sequences of data [18]. Hochreiter and Schmidhuber proposed the LSTM neural network [20] to remedy the vanishing gradient. The core unit in an LSTM consists of four main attributes, three gates and a cell state. Each of the gates control the information sent to and from the cell state. The input gate controls the update of the cell state while the forget gate evaluates and removes the irrelevant features of previous iterations. The last gate is the output gate which outputs the hidden state, a combination of the cell state and current input.

The different gates that update the cell state in the LSTM-unit allows the network to remember longer sequences of data. This is useful, e.g., in time series prediction where the current data point might depend on a longer sequence of previous data points. The LSTM network has been proven to be useful in time series analytic as well as in natural language processing and speech recognition [32] [40].

2.6.2 Support Vector Machines (SVM)

Another algorithm that is widely used in anomaly detection is the Support Vector Machine [14]. This algorithm is also a supervised learning method, though it does not require a uniform density of the different classes in the data set. Schölkopf et al. proposed a method for novelty detection using a One-Class SVM which does not require labeled data [35]. The principle behind novelty detection and anomaly detection is the same.

The SVM separates d-dimensional data of two different classes, +1 and -1, by a $(d-1)$-dimensional hyperplane while maximizing the margin between the hyperplane and the two classes of +1 and -1. Calculating the dividing hyperplane and simultaneously trying to maximize the margin between the classes leads to an optimization problem with a constraint. This approach is called linear SVM and an example of it can be seen in Figure 2.3.

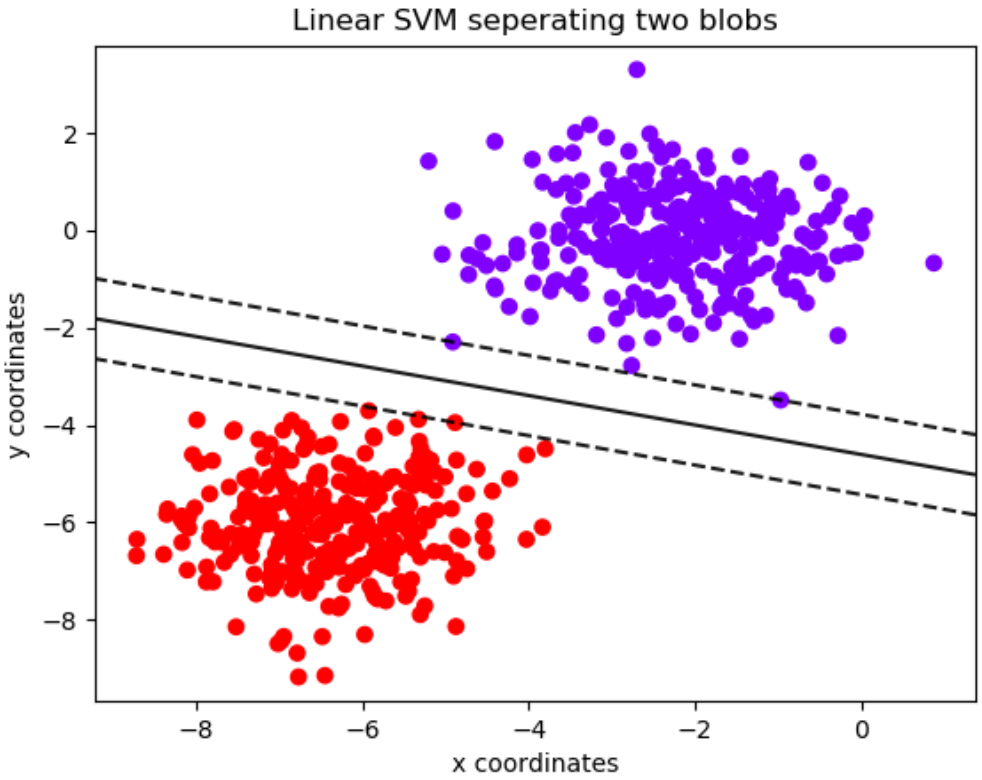


Figure 2.3: The figure shows a linear SVM separating binary class data in a 2 dimensional space with a line.

By finding a hyperplane defined as

$$\vec{x} \cdot \vec{\omega} + b = 0 \tag{2.9}$$

the goal is to take an input point, $\vec{x}$, of class +1 or -1 and classify it using

$$\begin{cases} \text{if} \quad \vec{\omega} \cdot \vec{x} + b \geq +1 \quad \text{then } +1 \\ \text{if} \quad \vec{\omega} \cdot \vec{x} + b \leq -1 \quad \text{then -1.} \end{cases} \tag{2.10}$$

By defining a constant ξ_i as

$$\xi_i = \begin{cases} 1 & \text{if } x_i \in +1 \\ -1 & \text{if } x_i \in -1 \end{cases} \tag{2.11}$$

Equation (2.10) can be rewritten as

$$\xi_i \left(\vec{\omega} \cdot \vec{x} + b\right) \geq 1. \tag{2.12}$$

The margin, μ, is defined as the distance between the two dotted lines in Figure 2.3 and can be calculated by maximizing the distance between the closest points $\vec{x}_{positive}$ and $\vec{x}_{negative}$ of the different classes.

$$\mu = (\vec{x}_{positive} - \vec{x}_{negative}) \frac{\vec{\omega}}{||\vec{\omega}||} \tag{2.13}$$

These points are called the support vectors. The hyperplanes that intersects them can be written as

$$\begin{cases} \vec{\omega} \cdot \vec{x}_{positive} + b = +1 \\ \vec{\omega} \cdot \vec{x}_{negative} + b = -1 \end{cases} \tag{2.14}$$

By combining Equation (2.13) and (2.14) the simple equation for the margin is given by

$$\mu = \frac{2}{||\vec{\omega}||} \tag{2.15}$$

To maximize the margin is equivalent to minimizing $||\vec{\omega}||$. For n data points the optimization problem can be formulated as:

$$\begin{cases} \text{minimize } ||\omega|| \\ \text{such that for } i = \{1, \ldots, n\} \quad \sigma\left(\vec{\omega} \cdot \vec{x}_i + b\right) = \xi_i \end{cases} \tag{2.16}$$

Where σ is the sign function and ξ_i is defined as in Equation (2.11). This optimization problem with constraints can be solved by using the Lagrange multiplier [30].

There are times when a linear SVM is not sufficient and the data can not be separated with a hyperplane. To solve this, one can replace the linear dot-product in the the Lagrangian function with kernel functions. A common kernel is the Gaussian Radial Basis Function, RBF, seen in Equation (2.17).

$$K(\vec{x}_j, \vec{x}_i) = e^{-\frac{|\vec{x}_j - \vec{x}_i|^2}{2\sigma^2}} = \left\{\gamma = -\frac{1}{2\sigma^2}\right\} = e^{-\gamma|\vec{x}_j - \vec{x}_i|^2} \tag{2.17}$$

Where the $\gamma = \frac{1}{2\sigma^2}$ is a parameter which takes the variance of the input data set into account. A smaller γ value allows larger variance in the input data to be mapped as similar. A large γ value only allow a small variance in the input data and will only map data points close

to each other as similar.

By using a kernel method, the data gets transformed into a higher dimension where is should be easier to separate the two classes by a hyperplane. An example of separating a non-linear problem using the Gaussian RBF is shown i Figure 2.4.

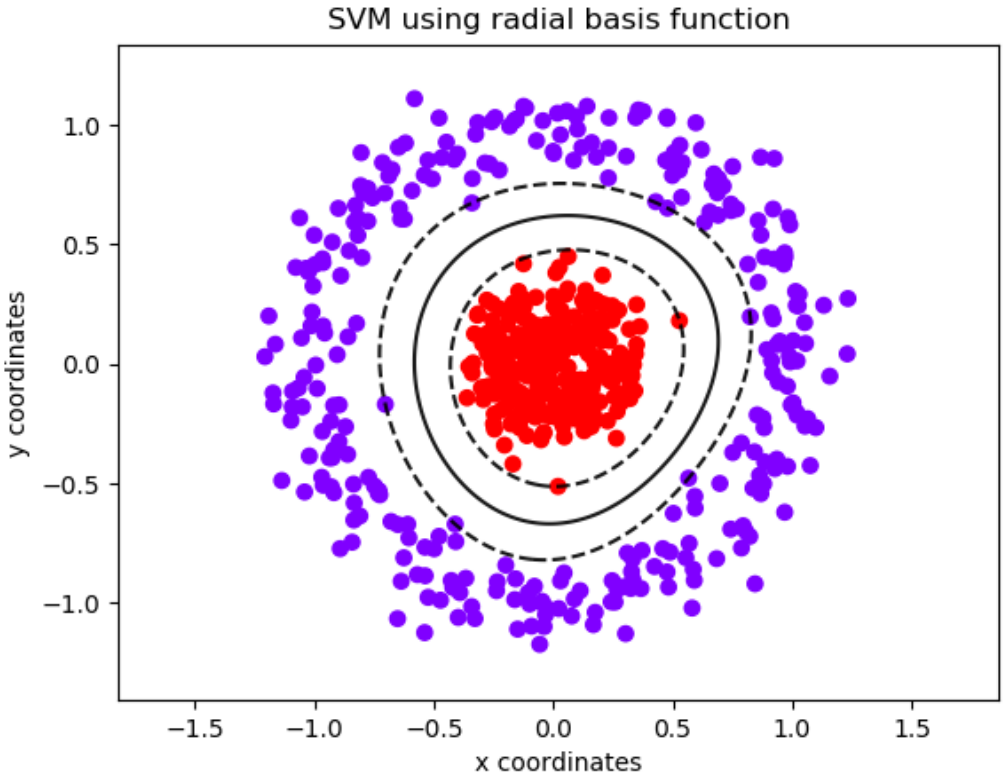


Figure 2.4: Figure showing the use of the radial basis function as kernel in an SVM to separate data with a non-linear decision boundary.

2.6.3 k-means Clustering

The k-means clustering algorithm is an unsupervised learning algorithm that places the unlabeled data into k clusters [26]. The main idea behind using this algorithm is finding clusters of data. By using the closest Euclidean distance, new data points can quickly be assigned to the different clusters. Binary classification for new points could be achieved by calculating the distance to the nearest cluster center depending on if the distance is higher or lower than a certain

threshold.

The steps in the algorithm are as follows. For m items, n dimensional dataset $X = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_m\} \in \mathbb{R}^n$, k cluster centroids $K = \{\vec{\kappa}_1, \vec{\kappa}_2, \ldots, \vec{\kappa}_k\} \in \mathbb{R}^n$ are randomly initialized. The algorithm then repeats two different procedures until it has converged.

In the first procedure, for each data point x_i, the squared Euclidean distance to each cluster centroid is calculated as in Equation (2.18). Each data point is then assigned to the nearest cluster K_l, where $l \in \{1, \ldots, k\}$.

$$\min \sum_{j=1}^{k} ||\vec{x}_i - \vec{\kappa}_j||^2 \tag{2.18}$$

The second step of the algorithm calculates the average position of all data points belonging to each specific cluster and places the cluster-centroid at the new position. This is repeated for each of the k clusters.

$$\vec{\kappa}_l = \frac{1}{t} \sum_{j=1}^{t} \vec{x}_j, \quad \forall \{\vec{x}_1, \ldots, \vec{x}_m\} \in K_l \tag{2.19}$$

Step one and two is repeated until the algorithm converges, meaning the clusters-centroid are no longer moving. In Figure 2.5, an example with $k = 3$ is shown where the black circles are the cluster centers after convergence.

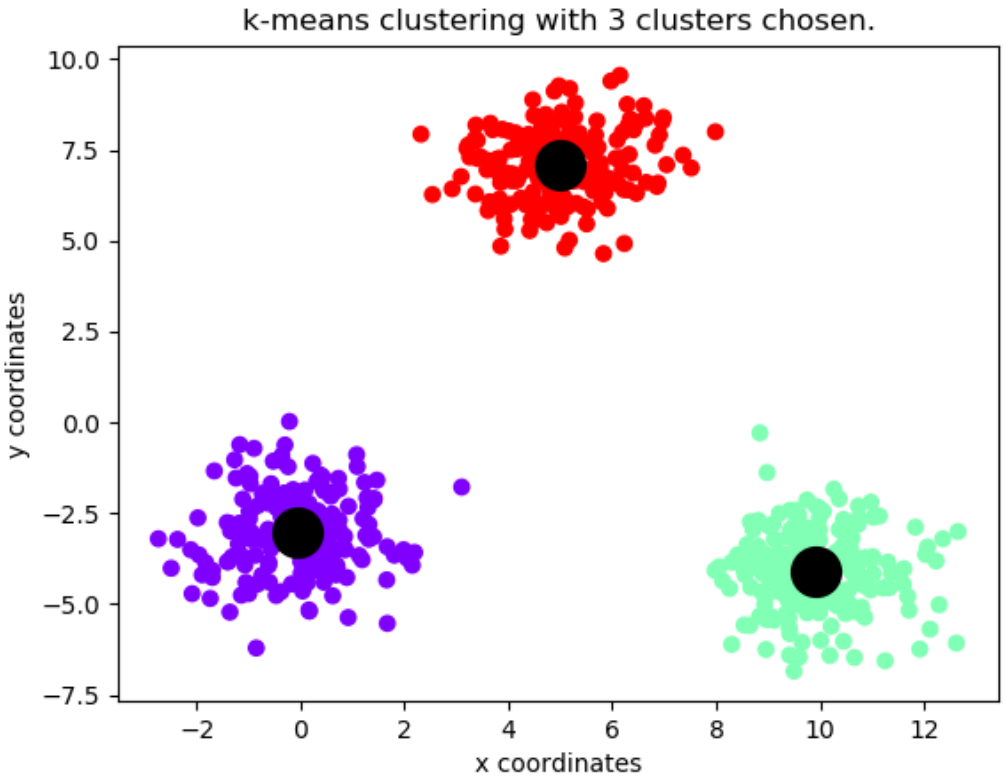


Figure 2.5: k-means clustering on three blobs of data is shown in this figure. The larger black points represent the three cluster centers calculated by the k-means clustering algorithm.

2.7 Metrics

This section explains the metrics that are used when measuring the accuracy of the machine learning models. The values used to calculate the metrics are given by a confusion matrix seen in Figure 2.6

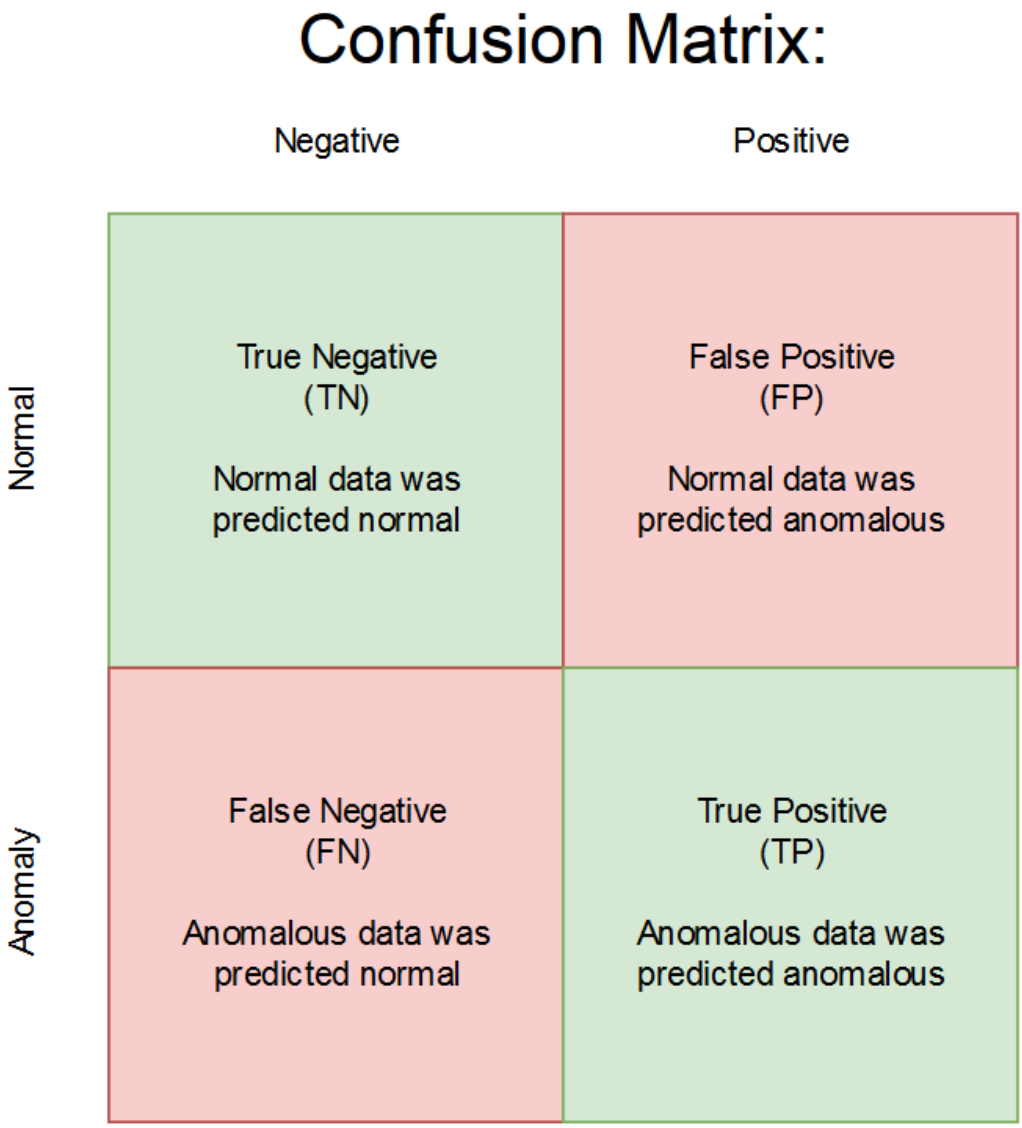


Figure 2.6: Figure displaying a confusion matrix and its components

The *true negatives* are the expected data points classified correctly while the *true positives* are the detections in a binary classification. Incorrect classification leads to *false negatives* and *false positives*, where the first are number of detections (positives) that are classified as normal (negatives) and the latter is when normal data is classified as detections. *Precision* is defined as a score of how many of the predicted positives were true positives.

$$precision = \frac{TP}{TP + FP}$$

Another important metric is the *recall*, which is a score of how many

of the total actual positives were found.

$$recall = \frac{TP}{TP + FN}$$

Combining the precision and recall score a widely used statistical measure of binary classification, the *F1-score*, can be defined. The F1-score is the harmonic mean of the precision and recall values.

$$F1\text{-}score = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

3

Methodology

This chapter describes the methods chosen for completing this project. The purpose is to motivate why certain methods were chosen. Section 3.1 describes what a lab for generating data requires. Section 3.2 explains how the data was generated, what features were extracted and motivates why. Section 3.4 shows the different pre-processing methods that were considered for this project. Section 3.5 motivates the choice of the machine learning algorithms for this project.

3.1 A network layout for generating and collecting traffic

To implement a live evaluation of machine learning algorithms a lab needs to be constructed. The lab should have one or more hosts performing some task over a network. The network should also contain one or more collection nodes that are not visible by the hosts on the network. These nodes never emit any traffic and their purpose is to collectively gather all the data in the network. The data from the collection nodes should be handled by a surveillance node which analyzes the data. There can be several surveillance nodes analyzing the data. If so, they must communicate separately from the analyzed network and have some algorithm for reaching consensus of the network analysis. Figure 3.1 shows a diagram over the general structure of a lab setup described in this section. This creates a lab where network traffic can be generated and collected.

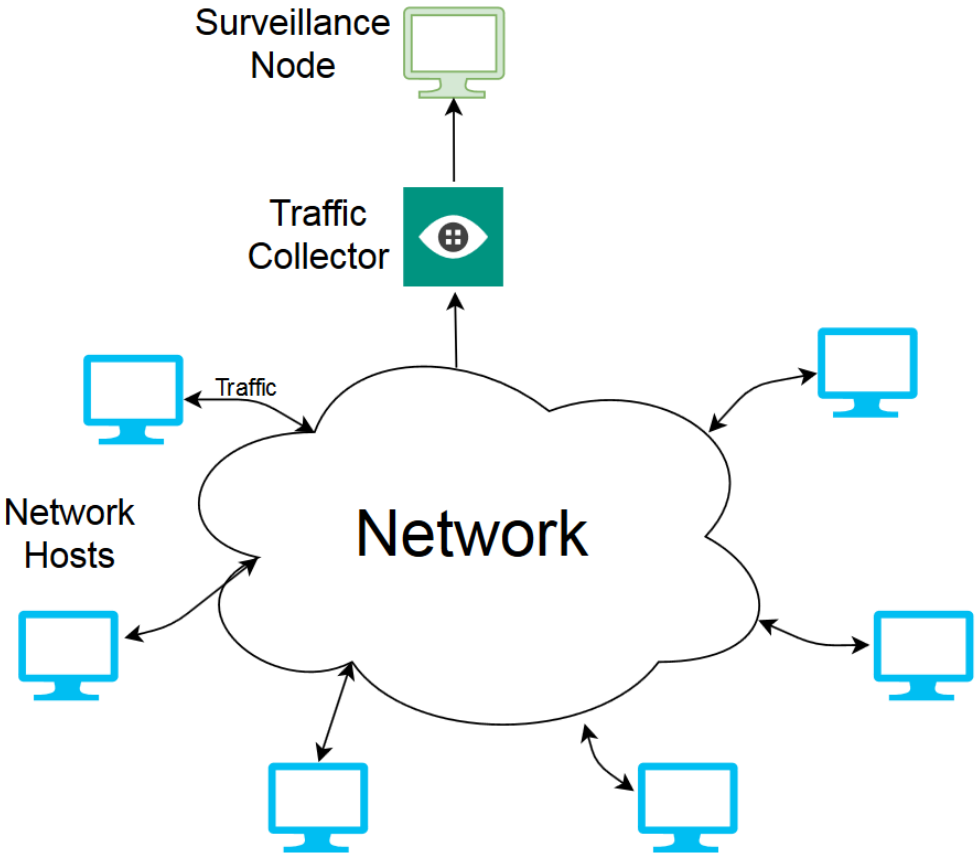


Figure 3.1: This figure displays the general structure of a network traffic collecting system using a single collection node.

3.2 Generating network traffic

Generating traffic between the nodes is an important part of the project. Traffic needs to be as realistic as possible and diverse enough to perform well as machine learning data. Several approaches were considered, most importantly whether to generate synthetic traffic between nodes or to have the nodes perform automated services towards each other such as ssh logins or web browsing. As the focus of the project is not to create a system of nodes that can simulate human traffic but rather to analyze computer traffic, the decision was made to run programs that can generate synthetic network traffic. In order to measure the traffic the libpcap tool was used together with a Python API.

The tools D-ITG, iperf and SourcesOnOff were tested for generating synthetic traffic over the network. These three were chosen among the various different tools because of their single purpose functionality and simple setup. During testing it was found that D-ITG was prone to crashing which made it a poor choice for longer sessions of generating network traffic. Iperf worked properly for long sessions but is built for bandwidth testing which resulted in it flooding the network with a high rate of packets per second. SourcesOnOff was able to produce a good variance of packet rates and was stable. Since SourcesOnOff performed best in the lab network this made it the most suitable of the three for this project.

It is arguable whether using synthetic network traffic increases or decreases the performance of the algorithms, as data collection and labeling is a large scientific topic in itself and the quality of the data is incredibly important for the performance of machine learning algorithms. The intent is that by using this self made data set a more accurate understanding of the network can be achieved, which in turn allows the algorithms to produce more accurate results. A problem with collecting data is the collecting time duration, which is correlated to the data quality. The KDD '99 data set contains 9 weeks of labeled network data which is more time than this project would allow to use for data generation. An improvement from using the KDD '99 dataset is that the pre-processing techniques relies on information extracted on a per packet basis from the network. This means that different pre-processing methods can be considered, tested and evaluated. When collecting the packets, a selected set of features from the IP header of each packet is saved for pre-processing. The features selected are based on the report by Benferhat et al. [9]. The duration feature is dropped since when capturing live traffic it is uncertain how long it will take to transmit the data. The chosen features are displayed in Table 3.1.

Feature	Description
Protocol	Transport layer protocol.
Source address	IP-address of the source node
Destination address	IP-address of the destination node
Source port	The logical port of the source node for the connection
Destination port	The logical port on the destination node for the connection.
Size	The size of the packet in bytes
Timestamp	The network packets timestamp
Flags	Status flag for the packet, if available.

Table 3.1: This table shows the features extracted from the IP headers of each captured network packet. The flag feature only contains SYN, ACK and FIN flags. If the transport protocol is UDP, the flags field is left empty.

These features contain enough information to map the network communication and is thus suitable for the machine learning algorithms to be able to detect network anomalies.

3.3 Generating anomalies with distinct features

Each anomaly is chosen and configured to be distinct from each other. A DoS attack causes an increase in network traffic, a node failure causes a decrease in network traffic and a port scan can be configured to not cause any increase or decrease in the network traffic. This made each of the anomalies proper for evaluation on a live network. This section shows how each anomaly was configured for this project. Frameworks for executing more complex anomalies were considered for this project, such as metasploit [29], but was ultimately discarded since there was a problem with automating the framework.

The type of DoS attack chosen was a SYN flood that sends a large amount of SYN packets to the targeted node. This attack was chosen since it is simple to implement and causes a large disturbance in the network flows which would make it noticeable in the data set.

The node failure simply needs to cause a node to not send or receive packets. This could be caused by manually unplugging the node from the network, but the solution should preferably be automated by software. The node could shut down the network interface or simply shut down the programs in charge of communicating over the network. The

chosen method for creating this anomaly was to kill the process handling the communicating program. Node failures are not complex and do not require more configuration than this. The important part is that the node failure causes the node to be inactive for enough time to be noticeable in the network. This anomaly should be harder to detect as normal network traffic varies and normal data rates in the network can be low. There can only be one node failing in the network at any given time, causing the network traffic to become unbalanced which makes the anomaly easier to detect.

The port scan was given a set of ports to scan using the TCP ACK flag in order to differentiate it from a SYN flood. A port scan can cause an increase in network traffic if a large number of packets are sent out in a short amount of time. To avoid this the port scan is configured to wait for a set time interval between all packets. This causes an increase in packets sent to unknown ports as well as an increase in ACK packets which should let the machine learning algorithm detect the port scan. The attacks are performed using a variety of Python and bash scripts, as well as the nmap tool [5].

3.4 Pre-processing raw network traffic into data sets

Capturing network traffic generates a lot of data and an important part of the project is to translate the raw network traffic into a data set that can later be fed into a machine learning algorithm. Three techniques were considered for pre-processing the data:

- Packet based numeric
- Time series
- Connections

These three techniques are described in the sections below. All three techniques are viable for creating a data set useful for machine learning. However, the time series pre-processing technique was the only one implemented in this project mainly due to the time constraint. The packet based numeric technique creates a very large data set which impacts the machine learning algorithms' learning time while the connections technique needs a system that handles unfinished connections

and out of order packets.

3.4.1 Packet Based Numeric

Packet based numeric is a relatively basic pre-processing technique. Packets are mostly left as is, but each feature is converted into an integer or a float to represent the original string. If the feature already has an integer or a float it is not converted. This retains most of the data from the original data set but will take a longer time for the algorithms to process compared to the other techniques, as more data points would be required. Since each packet would be interpreted as a singular data point with no regard to what packet came before, algorithms that use longer sequences of data points for classification would perform better. This is because it would be hard to detect an anomaly by looking at a single packet without knowing the context surrounding that packet.

Most of the features from the data collection set are trivial to convert into a numeric form. For example: if the protocol field says UDP, put 1, if it says TCP, put 2. The source port and destination port can effectively be merged into a single port feature. This is because when initiating a connection, the protocol on the initiating node selects a random source port for the connection. This random port is essentially useless when inspecting each packet individually. These two port features are merged into a single feature simply stating the service port. Another feature in the gathered network data that is not very useful in this context is the time feature. Instead of containing the time that each packet was captured, the time feature is converted to the time difference since last recorded packet. The first packet in the data set has a time feature of 0 since no packets were recorded before it.

3.4.2 Time Series

Each point in a time series takes a predefined time interval, for example 1 second, and each feature is divided into a set of feature counters. To give an example, if the protocol feature will only contain the values of TCP or UDP then the original feature is replaced with two new features, TCP counter and UDP counter. These counters increment

each time their specific feature appears during that second. The size feature is an exception, as it contains as many features as there are integers. Instead the size feature is the sum of all packet sizes that second.

This allows the algorithms to inspect each such time segment and be able to classify the segment based on an increase or decrease in certain features. Larger time intervals will decrease the number of data points that needs to be processed, allowing the machine learning algorithms to train faster. Larger time intervals can be detrimental as well since it obfuscates the variance in the data. The time intervals containing the network activity conceal some features as an anomalous packet can no longer be traced to the packets' individual features. This means that if two features are anomalous it is harder to see the connection between those features since there are no recollection of the features coming from the same packet. However, the time series representation can also amplify recurring anomalous features as they would add up in the chosen time interval process creating a more recognizable pattern. The features in each time interval is derived from the data extracted from the IP headers, seen in Table 3.1.

3.4.3 Connections

Connections are simple to define in a network using only TCP communication, since TCP has a definitive start and end. The proposed idea is that a TCP connection start would be defined as when the first SYN packet is transmitted and the end when the FIN + ACK is transmitted. A connection between two hosts can be identified by their IP-adresses and the port numbers. The individual packets are then summed up creating features that present how long the connection was and how much data was transmitted. Creating UDP connections is different since a UDP connection does not have a well defined end. The solution would be to have a time out when no new UDP packets using the same port numbers from the same host has been transmitted, indicating that connection is over.

3.5 Defining outliers and threshold for different Machine Learning algorithms

The methodology of how to define outliers for three different algorithms is described in this section. The algorithms presented are the LSTM neural network, the One-class SVM and the k-means algorithm. These specific algorithms are chosen since they are all able to employ unsupervised methods. The three algorithms have varying complexity and each one of the algorithms solves the detection of anomalies in a different way.

The main idea is that no labeled training data is available for the algorithms which means that only unsupervised and self-supervised methods can be applied on the problem. This requires a gathered network traffic data set that is assumed to contain no or an insignificant number of anomalies, where an anomaly is an outlier in the feature space. Assume that the set S is the entire data set of gathered network traffic and that S might contain a small subset A of malicious data points, $A \subset S$. Let N be the data points of S that are normal data, such that $N \subseteq S$. For a random data point $x \in S$, the probability of it being an anomaly is assumed to be small: $P\left(x \in A\right) = P\left(x \notin N\right) < \epsilon$, where ϵ is a small constant. In Figure 4.5, the threshold for the k-means algorithm (the squared Euclidean distance) needed for different values of ϵ between $0.10 \geq \epsilon \geq 0$ is shown.

For each algorithm, it will be assumed that a small number of data points are outliers to be able to determine a threshold. Furthermore, to define an anomalous data point it should be outside of the boundaries and the thresholds defined for each of the different algorithms which are explained below.

3.5.1 Long short-term memory neural network

The LSTM network as described in Section 2.6.1 was chosen due to a higher complexity and its ability to look at sequences of the time series. This algorithm applies a self-supervised approach which means that the algorithm is trained to predict the next point in the time series. By doing so it take a sequence of previous time series data points and

use the subsequent point as target output for error estimates. Since the LSTM is a type RNN it stores information from previous data points in the time series and its task can be described as in equation (3.1).

$$\text{for} \quad \{\vec{x}(i-h), \ldots, \vec{x}(i)\} \quad \text{predict} \quad \vec{x}(i+1) \tag{3.1}$$

After the LSTM network is trained on the network traffic training set, it is evaluated by checking the root mean square error ($RMSE$) for each of the data points in the test set. Assume that $\xi(i)$ is the predicted data point, and $x(i)$ is the actual data point, then the root mean square is

$$RMSE = \sqrt{\left((\xi_1(i) - x_1(i))^2 + \cdots + (\xi_d(i) - x_d(i))^2\right)}, \tag{3.2}$$

where d is the dimension of the data set. Assuming that the data contains a few anomalous points a threshold that differentiates normal network traffic data and anomalous data is calculated based on $RMSE$. When introducing a new data point $\tilde{x}$, it is classified as anomalous or normal based on if the $RMSE$, of the prediction from the LSTM and the actual point, is less or greater than the threshold as seen in equation (3.3).

$$\tilde{x} = \begin{cases} \text{normal} & \text{if } RMSE \leq T \\ \text{anomalous} & \text{if } RMSE > T \end{cases} \tag{3.3}$$

3.5.2 SVM

The standard SVM described in section 2.6.2 takes two classes and tries to separate them. This cause a problem since the data set in this thesis only consist of a single class, namely what is assumed to be normal network traffic data. The One-class SVM however, handles the one class problem by calculating a hypersphere around a part of the data it is fed. All the data points inside the hypersphere are classified as normal data point and all the data points outside the hypersphere are classified as outliers or anomalies. This makes this algorithm a suitable choice for anomaly detection when only unlabeled, and what

is assumed to be normal, data is available.

The hyper parameters to be tuned for the One-class SVM are γ, which is explained in Section 2.6.2, and ν. The ν parameter governs the upper bound on the fraction of margin errors and a lower bound of the fraction of support vectors relative to the total number of training examples. A low ν value penalizes data points that are outside the hypersphere while a higher ν value generalizes better by accepting more outliers. By adjusting ν, a part of the data ϵ that might be malicious can be excluded. There is no general solution to set the values of γ and ν and in order to find proper values for the hyper parameters a range of different values needs to be tested and evaluated. The One-class SVM that yields the best results will be the one used in the live performance evaluation.

3.5.3 k-means clustering

As mentioned in section 2.6.3 the k-means clustering is an unsupervised method that tries to define clusters in the data. By using this algorithm to find clusters of normal network traffic data, the intention is to find a set of clusters such that they represent the normal network traffic. The definition of what is normal and abnormal network traffic is based on the squared Euclidean distance from the nearest cluster center as in equation (3.4).

$$d(x) = \min \sum_{i=1}^{k} ||\vec{c_i} - \vec{x}||^2 \tag{3.4}$$

Where d is the squared Euclidean distance, c_i is the position of the cluster centers and $\vec{x}$ is a data point. To be able to decide if a new point is anomalous or normal a threshold for the maximum distance squared to nearest cluster is set.

Of all network traffic data points in S, the threshold, T, is chosen such that a subset $N \subseteq S$ will be classified as normal. Then T is chosen as in Equation (3.5).

$$1 - P\left(d(x) < T\right) < \epsilon, \quad \forall x \in X \tag{3.5}$$

Where ϵ is the allowed percentage of the data point in S that might contain anomalous data.

4

Experimental Set-up

This chapter describes how the experiments that were used to complete the project were set up. First the lab was set up as described in section 3.1. The lab is used to generate synthetic network traffic between the nodes using the selected traffic generation programs. The network traffic is gathered to be processed into data that is comprehensible to the machine learning algorithms. When the algorithm has achieved a classification of the data the result will be a measure of how accurately it detected anomalies and normal network traffic. Then the procedure is restarted at the machine learning stage and tweaked according to the evaluated results until the machine learning algorithm is able to perform well. When an algorithm is deemed to give satisfactory results a new algorithm is selected and the same procedure is restarted. A flow chart over this process can be seen in Figure 4.1.

4.1 Lab Setup

The lab consists of five computers and a switch. Four of the computers are nodes communicating with each other over the network while the fifth will be the supervising node trying to detect anomalies. All computers are connected to a switch that forwards all traffic to the supervising node. Figure 4.2 shows a diagram over the lab setup. Nodes 1-4 in Figure 4.2 are running software communicating over the network. The surveillance node is running the data collecting software together with the machine learning algorithms. The nodes on the network are not running any type of firewall or other security measures but are running with standard raspbian settings from 2019-01-25. Table 4.1 shows what protocols are used by each open port.

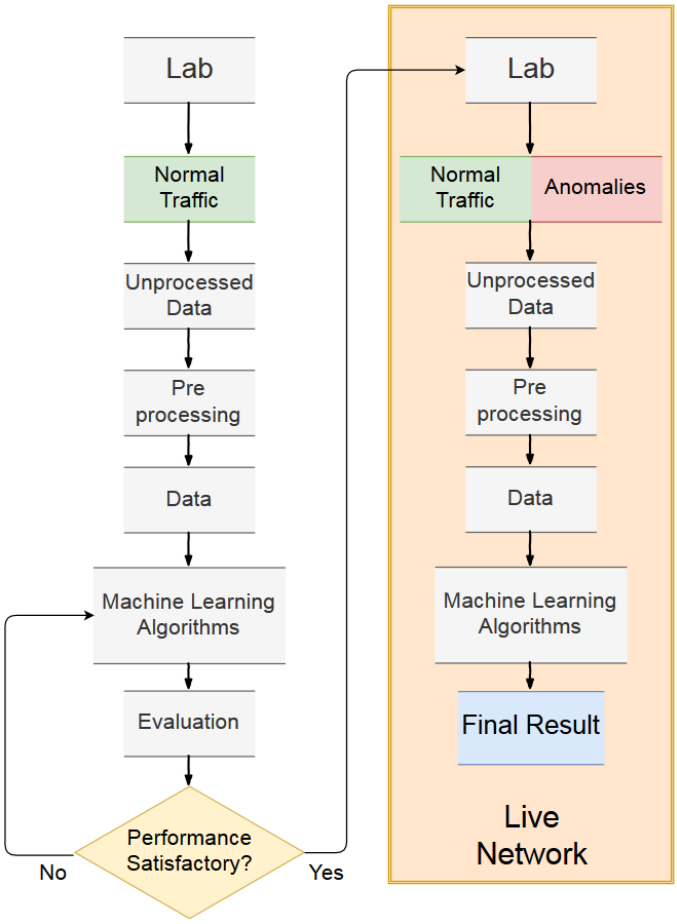


Figure 4.1: Chart giving an overview of the workflow. The lab creates normal network traffic that is to be processed into data that can be understood by the machine learning algorithm. The algorithm is then evaluated depending on how well it performs. The process is tweaked until performance is satisfactory. Once the algorithm can correctly classify normal data it is evaluated on a live network with anomalous traffic present in the network

Protocol	**Ports**
TCP	9555, 9271, 5201
UDP	8345, 9955, 5633

Table 4.1: Table showing which ports the protocols TCP & UDP are able to send to.

4.2 Generating network traffic in the lab

As internet network traffic varies greatly throughout the day, from up to 95% TCP traffic to 75% UDP traffic [25], the traffic is made to have around 10 - 15% of the traffic generated to be UDP. This percentage is chosen in order to have a noticeable difference in the number of

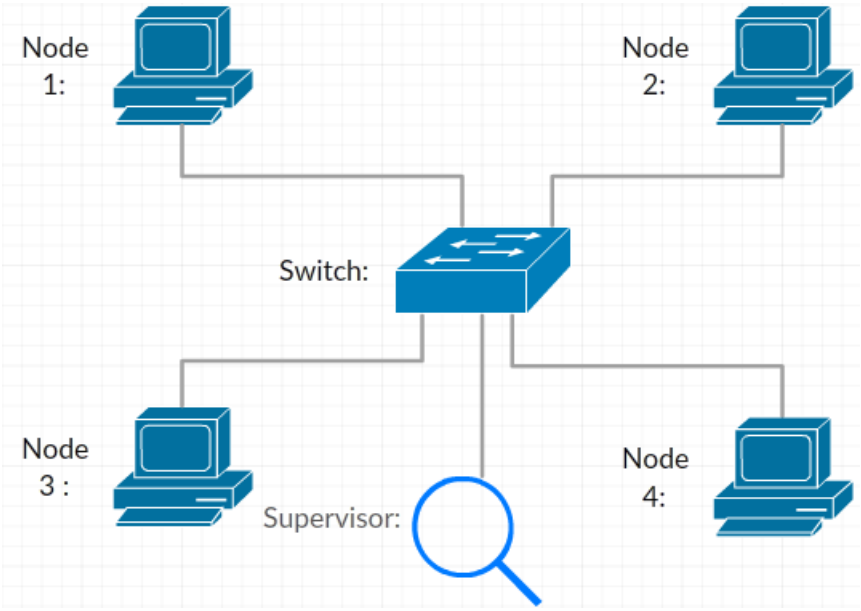


Figure 4.2: The network topology of the lab showing how the computers are connected to each other. Node 1-4 are regular computers communicating over the network. The supervising node is the node collecting traffic and running the machine learning algorithms on the network data.

Protocol	Percentage of total network traffic
TCP	89%
UDP	11%

Table 4.2: A table showing an example of how much of the network traffic is UDP and how much is TCP. This traffic varies slightly due to induced randomness.

packets using the different protocols. Table 4.2 shows an example of the proportions of TCP and UDP traffic in the network. In order to better imitate real systems a set of ports are selected to receive TCP traffic and another set of ports are selected to receive UDP traffic. The TCP ports selected are 5201, 9271, 9555 and the UDP ports are 5633, 8345, 9955. There are no real services running on the nodes or on the network, only the synthetic traffic between the SourcesOnOff services that is specified in this report. Each node has a pattern on how it should select the receiving node when sending data. This pattern is displayed in Table 4.3. Note that no node should ever send any traffic to itself. A graph showing the network traffic can be seen in Figure 4.3.

When creating anomalies, a python script is used to automatically per-

Receiving Node / Sending Node	Node 1	Node 2	Node 3	Node 4
Node 1	0%	15%	10%	75%
Node 2	15%	0%	20%	65%
Node 3	15%	10%	0%	75%
Node 4	20%	40%	40%	0%

Table 4.3: Table showing the chance of each node selecting a specific node as a recipient.

form the network anomalies. The anomaly creation script is running on one of the four communicating nodes in the network. The script has an equal chance of running a node failure, SYN flood or a port scan. To generate a node failure the script kills the SourcesOnOff server and respond scripts for a given interval of time and then resumes the SourcesOnOff program. The SYN flood selects a random node and a random port that is running a TCP service and sends between 20,000 and 50,000 SYN packets to that port. The receiving node for a SYN flood is selected randomly and does not follow the probabilities in Table 4.3. When performing a SYN flood the attacking node spoofs an IP-address of one of the nodes in the network. The port scan performs an ACK scan, which is to send TCP ACK packets to different ports. This is to differentiate it more from a SYN flood. The script selects an interval of 100 to 10,000 ports on the host, which is selected the same way as in the SYN flood attack, and sends packets to the selected ports with a time delay of 0.001 seconds between packets. This delay causes the port scan to not increase network congestion and makes it harder to recognize. When a node starts an anomaly it saves what time the anomaly started, what time it ended and what type of anomaly was performed. A graph showing the network traffic but with the anomalies marked can be seen in Figure 4.4.

4.3 Processing the collected network traffic

The lab uses SourcesOnOff to produce traffic between nodes 1-4. This traffic is then forwarded to a supervising node that filtered the traffic and turns it into useful data. The supervising node does this by using libpcap to check the IP headers for information as mentioned in section 3.2.

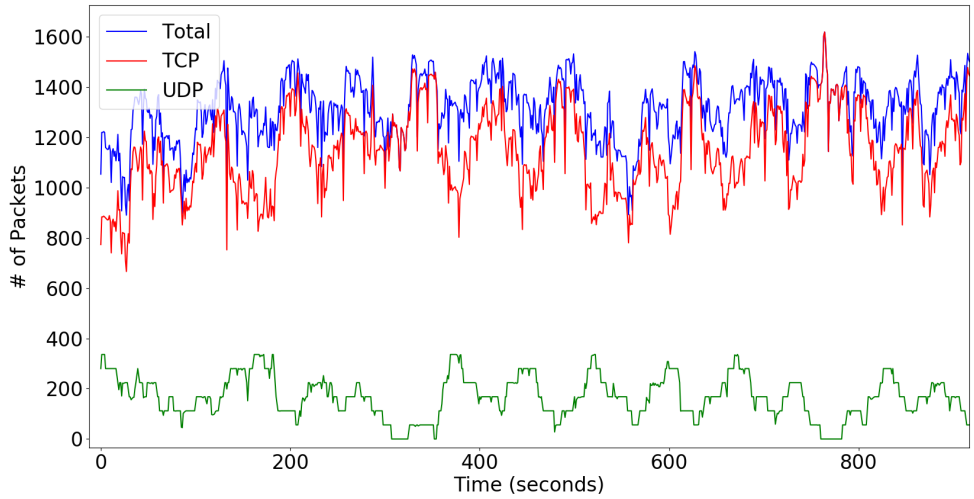


Figure 4.3: Graph showing the amount of packets sent over time. The red line shows how many TCP packets are transmitted per second. The green line shows how many UDP packets are transmitted per second. The blue line shows how many packets per second are transmitted in total. There is no anomalous traffic displayed in the graph.

This data set of features is then labeled using the anomaly information from the node that performed the network anomaly generation. The data set is labeled as either 'SYN Flood', 'Port Scan', 'Node failure' or 'Normal' corresponding to the start and end times of the different anomalies. If the time stamp is not inside any anomaly start and end time interval the label is set to 'Normal'. This labeling is used to measure the machine learning performance. When a SYN flood, port scan or node failure is detected, the whole interval of the data is marked as the corresponding anomaly. This means that the otherwise normal data that is transmitted during the anomaly duration is also labeled as anomalous. This labeling is done to measure how efficient the machine learning algorithms are at identifying the different anomalies.

The gathered network data is processed into one second time series. The features of the time series bins can be seen in Table 4.4. The features are derived from the network topology of the network and the services running on the network. When the captured traffic is transformed into time series as shown in section 3.4.2 the size of data sets used for training is reduced by 99 %.

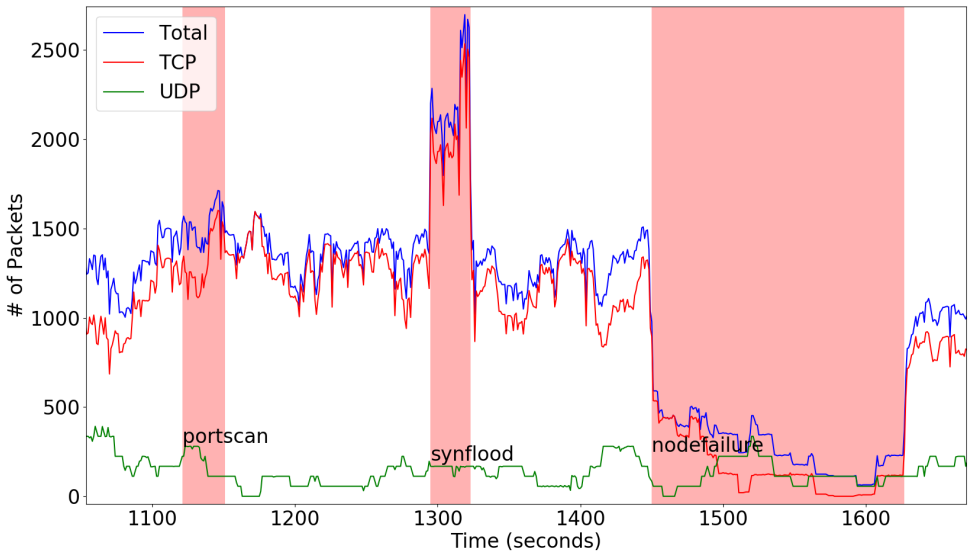


Figure 4.4: This graph shows a relatively small interval of network traffic as in Figure 4.3 but with anomalous network traffic mixed in. The anomalous traffic intervals are marked with their respective labels. The anomalies are from left to right: port scan, SYN flood, node failure.

Before using the machine learning algorithms the data set is centered around origin, such that no specific feature is too decisive. The data set is scaled using standardization, seen in Equation (4.1), which transform the input data to have zero mean and unit variance.

$$x_{standardized} = \frac{x_{input} - \mu}{\sigma} \tag{4.1}$$

The scaling is done on the normal network traffic dataset and saved. This scaling is then used in the live implementation.

4.4 Determine Hyper Parameters and Thresholds for the Machine Learning Algorithms

The machine learning algorithms are trained using a data set containing only normal network traffic data. For the k-means algorithm and the LSTM network suitable thresholds are calculated such that they exclude part of the data set that might contain anomalous data. For the One-class SVM different values of the hyper parameters ν and γ

Bin	Description
TCP	# TCP packets
UDP	# UDP packets
Host 1	# packets sent from node 1
Host 2	# packets sent from node 2
Host 3	# packets sent from node 3
Host 4	# packets sent from node 4
9555	# packets to/from port 9555
9271	# packets to/from port 9271
5201	# packets to/from port 5201
8345	# packets to/from port 8345
9955	# packets to/from port 9955
5633	# packets to/from port 5633
Other port	# packets to unknown port
Total size	Sum of all packet sizes for this second
SYN	# packets with SYN flag set
ACK	# packets with ACK flag set
FIN	# packets with FIN flag set

Table 4.4: Table shows the bins extracted from the features in table 3.1 for a time series analysis.

are iterated until a suitable combination is found that also excludes a small portion of possible malicious data points.

4.4.1 k-means Clustering

The plot in Figure 4.5 shows the distribution of the squared Euclidean distance between the network traffic data points and the cluster centers. The plot in Figure 4.6 shows the threshold needed to include a specific portion of the normal network traffic data. The value of T is calculated by evaluating a range of difference percentages, ϵ, between 10 and 0%. Where an ϵ of 10% would mean that the network traffic data might contain 10% of malicious data, or inversely consist of 90% normal network traffic data. The k-means clustering algorithm is evaluated multiple times since it is not guaranteed to find the global minima.

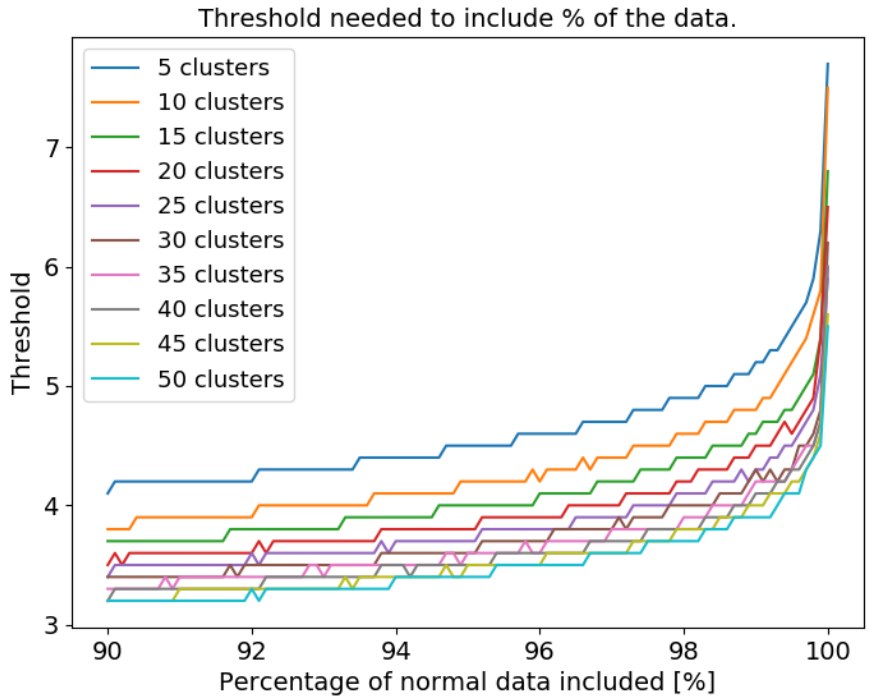


Figure 4.5: Figure shows the threshold, squared Euclidean distance, needed to classify a specific percentage of the network traffic data as normal.

By looking at Figure 4.5 it is clear that the threshold needs to be increased by a lot to be able to classify the last percent of the normal network traffic data. Including 100% of the data gives a exponentially larger threshold which can lead to larger amount of anomalous data being classified as normal. The threshold needed to label part of the data set as normal increases significantly faster when $k < 25$ clusters. Therefore choosing between $30 \leq k \leq 50$ clusters centers is of interest, since a smaller threshold means that the clusters represent the data better and a smaller number of cluster centers means that classifying new data points are faster. Because of this using k=40 clusters centers is chosen.

For the 40 clusters, the distribution of the Euclidean distance squared to the nearest cluster is shown in Figure 4.6 together with the value that separates the last 0.5% of the normal traffic data. This value is the threshold chosen for the 40 clusters that is going to be implemented in the live network.

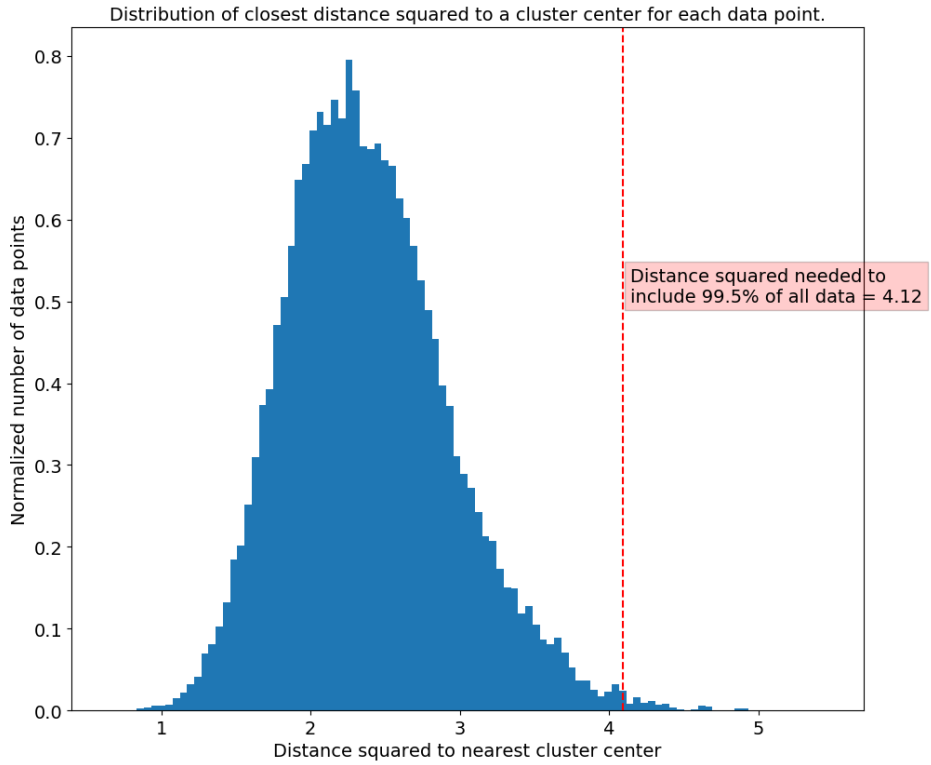


Figure 4.6: This figure show the distribution of the distance squared to the final clusters. In this particular run a threshold of 4.12 was chosen to include 99.5% of the normal network traffic.

4.4.2 One-class SVM

Several One-class SVMs were trained and tested on the gathered normal data. When training a One-class SVM the value of ν needs to be in the range (0, 1] and γ needs to be kept low for the radial basis function to create a greater decision boundary. The figures shows the false accuracy when the One-class SVMs are evaluated on the test set using different ν and γ values. Figure 4.7 displays a graph of the accuracy for different ν and γ values. Figure 4.8 displays the same graph but zoomed in on the values that gave the highest accuracy. The choice to include 99.9% of the data was made in order hinder the algorithms from training on any possible outliers in the data set.

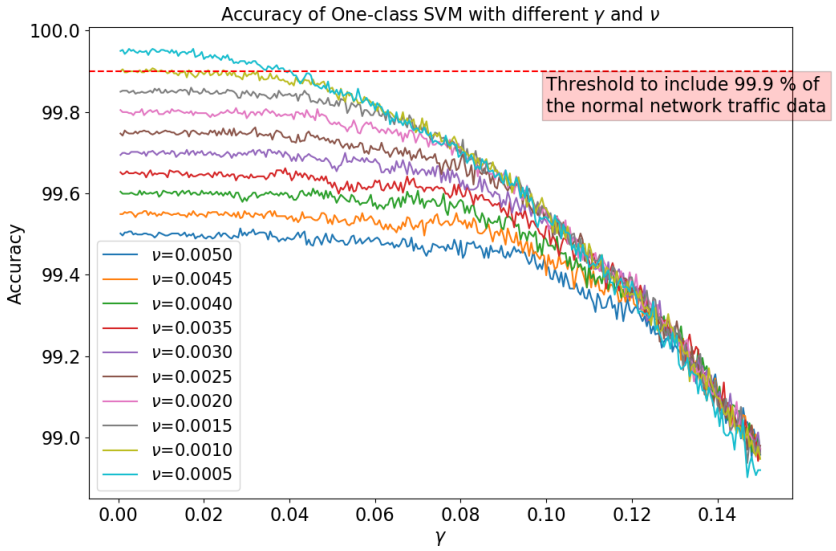


Figure 4.7: Testing different values of γ and ν on normal traffic data. The threshold is set to include 99.9% of the normal traffic data. Lower values of γ and ν increases the percentage of network traffic classified as normal.

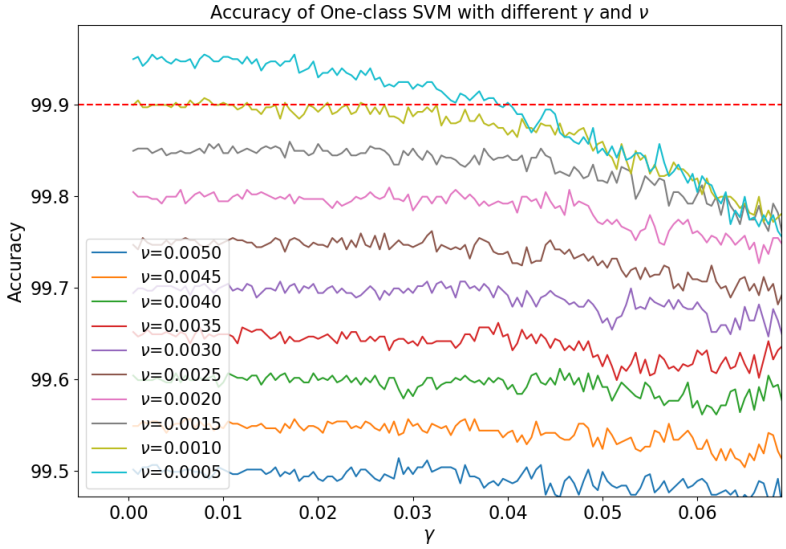


Figure 4.8: Zoomed in version of figure 4.7. The threshold is set to classy 99.9% of the network traffic data as normal, which is reached for a $\nu \leq 0.001$ and $\gamma \leq 0.007$.

From these results on collected normal data the choice was made to create a One-class SVM with a ν value of 0.001 and a γ value of 0.04. This was because the One-class SVM needed to be as close to the chosen accuracy threshold as possible in order to not create a too large hypersphere.

4.4.3 LSTM

The LSTM network is a type of Recurrent Neural Network. This means that it requires a network structure, a loss function, an optimizer and a learning rate. The learning rate is set to 0.01, the loss function is the Mean Squared Error (MSE) and the batch size is set to 64. The optimizer used is Adam [24]. Three different network architectures are tested, with varying numbers of hidden LSTM-layers and neurons, as seen in Table 4.5.

Model \ Layers							
3_{LSTM} 2_{DENSE}	LSTM_{64}	LSTM_{128}	LSTM_{64}	DENSE_{64}	DENSE_{17}		
4_{LSTM} 2_{DENSE}	LSTM_{64}	LSTM_{128}	LSTM_{128}	LSTM_{64}	DENSE_{64}	DENSE 17	
5_{LSTM} 2_{DENSE}	LSTM_{64}	LSTM_{128}	LSTM_{256}	LSTM_{128}	LSTM_{64}	DENSE_{64}	DENSE_{17}

Table 4.5: Different network structures tested for the LSTM-algorithm where LSTM_{64} meaning a hidden layer with 64 neurons.

For the LSTM layers a time sequence of 60 seconds is being fed into the algorithm. Furthermore the network data set is split into three parts: training, validation and test set. Where the algorithm is trained on the training set, the validation set is used to monitor the training such that the model does not overfit on the training data. The algorithm is then tested and evaluated on the test set.

The threshold is calculated for different values off $(1 - \epsilon)$ on the test set, where the $\epsilon = P(x \in A)$ for any point $x \in S$. The fact that the threshold is calculated on the test set is to reduce the risk of the network being overfitted on the training set, thus yielding a smaller threshold. In Figure 4.9 the threshold needed to classify between 90% and 100%, of the test set, as normal network traffic data is plotted. The goal is to include a large part (>99%) of the set S. The curves of both the 4_{LSTM} 2_{DENSE}- and 3_{LSTM} 2_{DENSE}-models are showing that they are able to include more data with the same threshold than the 5_{LSTM} 2_{DENSE}-model. Because of this, 4_{LSTM} 2_{DENSE}-model is the one being chosen when implementing it live.

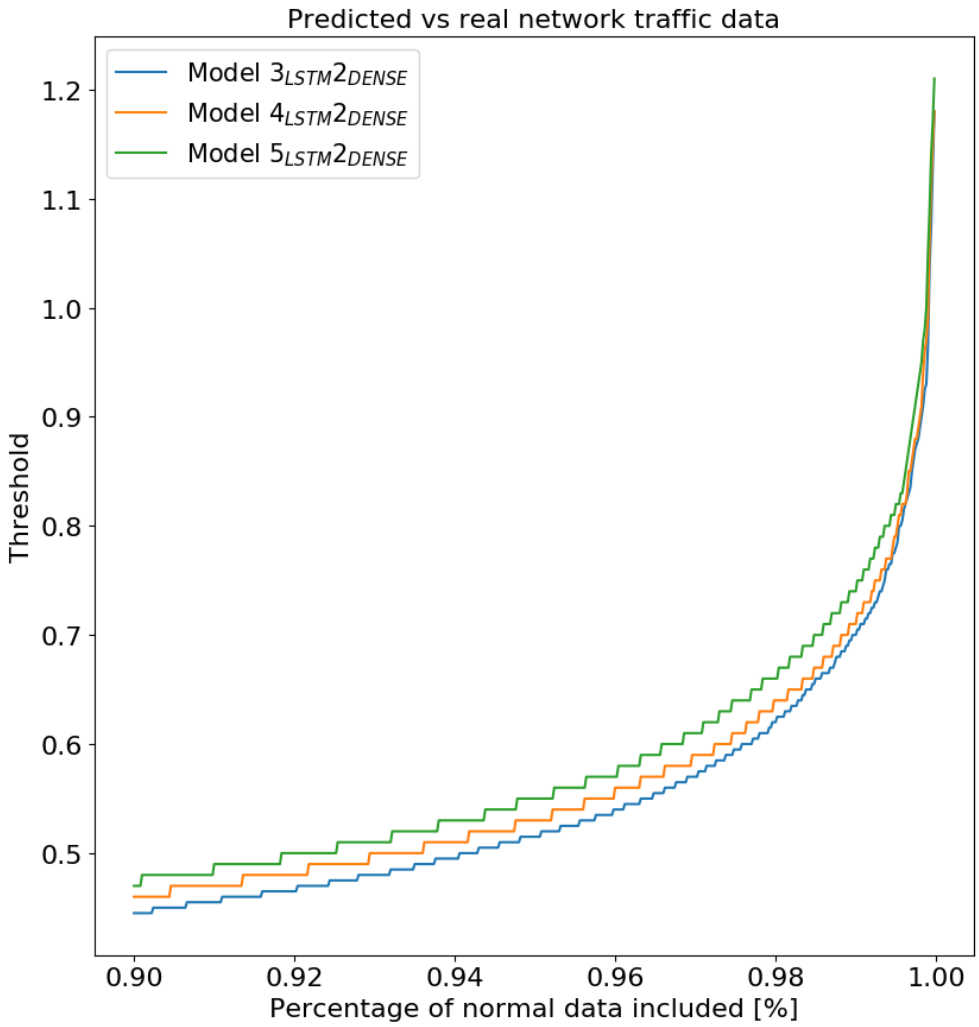


Figure 4.9: This figure shows the $RMSE$ threshold needed to classify a specific percentage of the network traffic data as normal. The three different network architectures in Table 4.5 is tested.

In Figure 4.10 the distribution of $RMSE$ for each data point in the test set are plotted. The red vertical line denotes the threshold needed to include 99.5% of the data points in the test set.

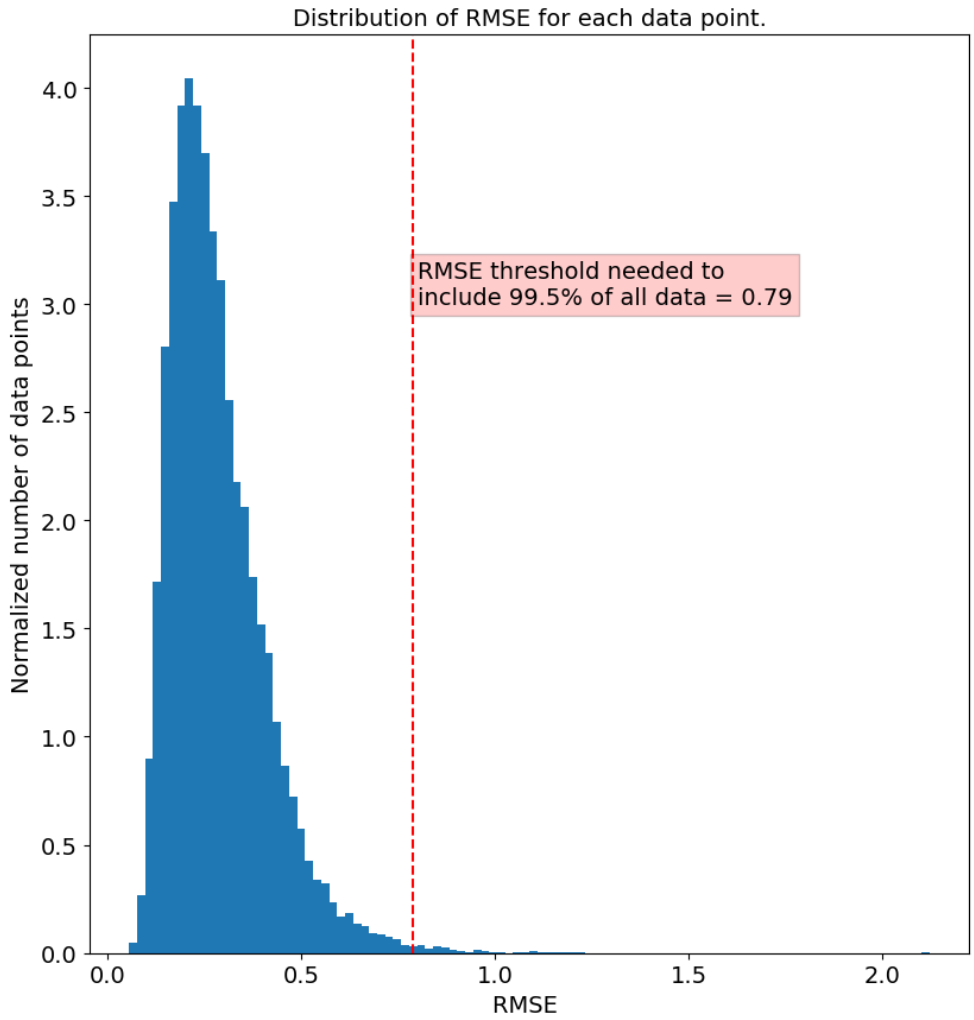


Figure 4.10: This figure shows the distribution of $RMSE$ of the predicted and actual data. The amount of data to the left of the red vertical axis is 99.5%.

5
Results

For evaluating the algorithms the terms false positive, false negative, true positive and true negative are used. False positive means normal data was classified as anomalous. False negative means that anomalous data was classified as normal. True positive means anomalous data was classified as anomalous. True negative means that normal data was classified as normal. These values are used when calculating the F1-score, precision and recall values.

5.1 Baseline

The result of the chosen algorithms are compared to the results of the same algorithms found in different papers. The paper chosen for comparison for the k-means clustering algorithm is "Research of K-MEANS Algorithm Based on Information Entropy in Anomaly Detection" by L. Han [19]. The LSTM neural network is compared to "Long Short Term Memory Recurrent Neural Network Classifier for Intrusion Detection" by Kim et al. [23]. Finally the One-class SVM results are compared "An Anomaly Detection Model Based on One-Class SVM to Detect Network Intrusions" by Zhang et al. [45]

5.2 k-means Clustering

In this section the result from the k-means clustering algorithm is presented. From the tests in Section 4.4.1, a cluster size of 40 was chosen. The results were gathered over a 4 hours long live evaluation resulting in 15,540 time series data points. Table 5.1 shows how well the algorithm was able to classify the different classes. Table 5.2 shows how well the k-means algorithm performed on live data compared to the results shown from Han's research [19]. Figure 5.1 shows the resulting confusion matrix. The overall accuracy of the k-means clustering

algorithm was 98.12%.

	Correct prediction	Total number	Percentage
Normal	10101	10303	98.04%
Syn flood	1368	1370	99.85%
Port scan	2288	2321	98.58%
Node failure	1492	1546	96.51%

Table 5.1: Table showing the classification statistics of the *k*-means clustering algorithm

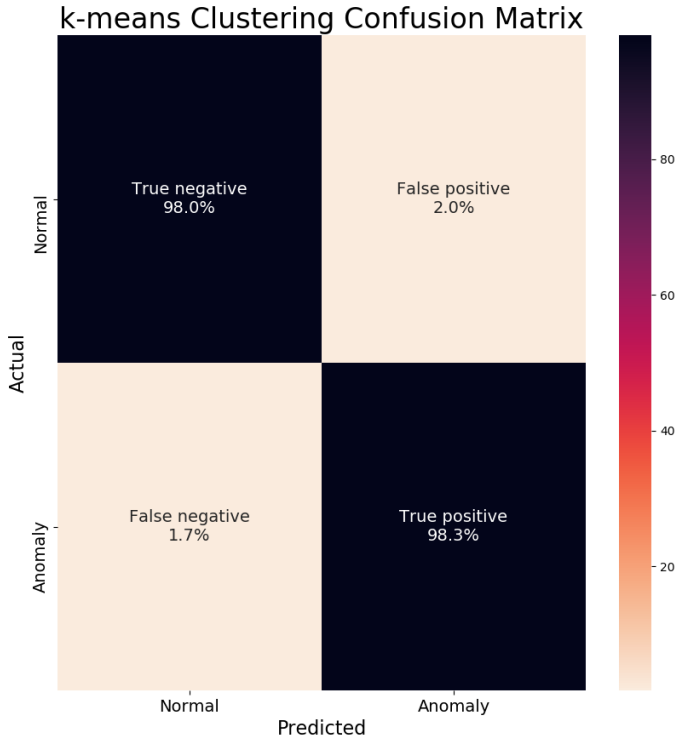


Figure 5.1: This figure shows the confusion matrix of the k-means after evaluating a live network with synthetic attacks.

	Target score	**Achieved score**	**Difference**
F1-score	0.9335	0.9725	0.0390
Precision score	0.9584	0.9622	0.0038
Recall score	0.9098	0.9830	0.0732

Table 5.2: Table comparing reported results from research by L. Han [19] to the results produced by the *k*-means algorithm of this project.

5.3 LSTM

In this section the results from the LSTM is presented. From the results presented in Section 4.4.3, the 4_{LSTM} 2_{DENSE}-model was selected with a threshold of 0.79 to be evaluated on the live network. The results were gathered for 7.71 hours providing 27,750 evaluated data points. Table 5.3 shows how well the algorithm was able to classify the different classes. Table 5.4 shows how well the algorithm performed on live data compared to the research by Kim et al. [23]. Figure 5.2 shows the resulting confusion matrix. The overall accuracy of the LSTM algorithm was 96.48%.

	Correct prediction	Total number	Percentage
Normal	17266	18030	95.76%
Syn flood	2667	2714	98.27%
Port scan	4634	4728	98.01%
Node failure	2200	2277	96.62%

Table 5.3: Table showing the classification statistics of the LSTM algorithm

	Target score	**Achieved score**	**Difference**
F1-score	0.9411	0.9511	0.01
Precision score	0.9866	0.9256	-0.061
Recall score	0.8996	0.9781	0.0785

Table 5.4: Table comparing reported results from research by Kim et al. [23] to the results produced by the LSTM in this project.

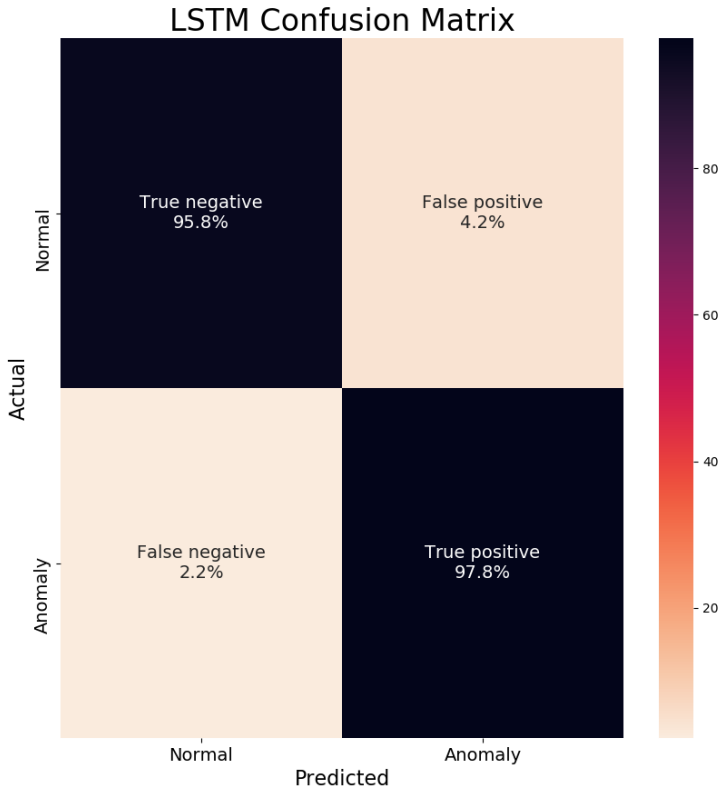


Figure 5.2: This figure shows the confusion matrix of the LSTM network when presented to new data is shown.

5.4 One-class Support Vector Machine

In this section the results of the One-class SVM is presented. From the results shown in Section 4.4.2, the One-class SVM with a ν value of 0.001 and a γ value of 0.04 was chosen. The results were gathered for 4.26 hours providing 15,349 evaluated data points. Table 5.5 shows how well the algorithm was able to classify the different classes. Table 5.6 shows how well the algorithm performed on live data compared to the research by Zhang et al. [45]. Figure 5.3 shows the resulting confusion matrix. The overall accuracy of the One-class SVM algorithm was 98.6%.

	Correct prediction	Total amount	Percentage
Normal	10195	10302	98.96%
Syn flood	1593	1597	99.75%
Port scan	1860	1893	98.26%
Node failure	1482	1557	95.18%

Table 5.5: Table showing the classification statistics of the One-class SVM

	Target score	**Achieved score**	**Difference**
F1-score	0.9518	0.9783	0.0265
Precision score	0.9903	0.9788	-0.0115
Recall score	0.9161	0.9778	0.0617

Table 5.6: Table comparing reported results from Zhang et al. [45] to the results produced by the One-class SVM in this project.

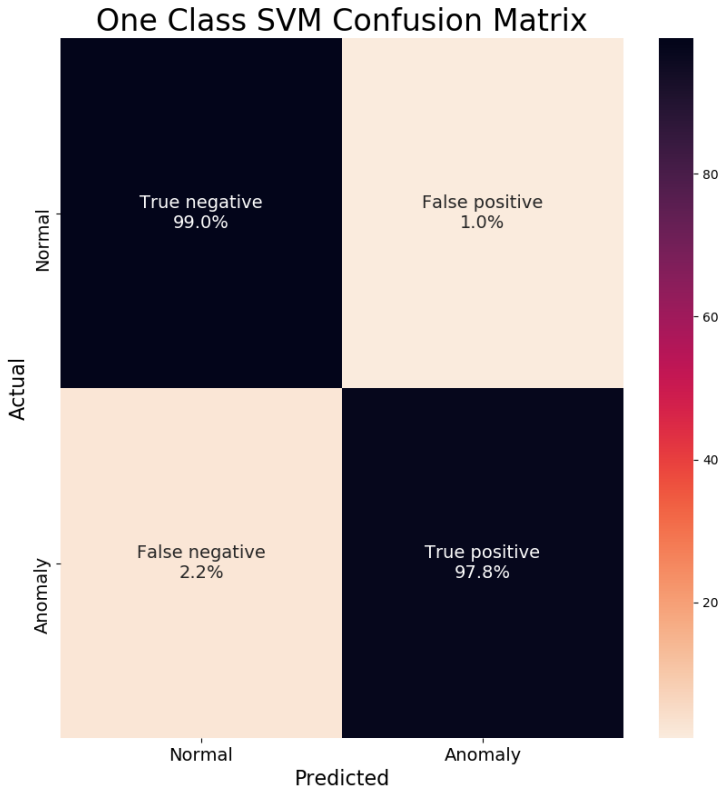


Figure 5.3: This figure shows the confusion matrix of the SVM when evaluated on a live network

6

Discussion

In this chapter we are discussing the chosen methodology, the results of the different machine learning techniques and how their performance correlates to the experimental lab setup.

6.1 Results - good recall score at the expense of precision score

By looking at the results and the comparison with similar algorithms on the KDD '99 data set there are two distinctive differences. Two of the three algorithms have a higher precision when trained on KDD '99 while all algorithms trained in this thesis have a significantly higher recall score. The decrease in precision means that the algorithms trained in this project has a slightly larger risk of mistaking normal traffic as anomalous. The bigger difference in the recall score however means that the algorithms trained in this project found a lot more anomalies than the ones trained using KDD '99. The resulting positive difference in the F1-score for all three algorithms indicates that the trade off between the slight decrease in precision and the increase in recall results in a more accurate algorithm. This can be attributed to the accuracy thresholds when tuning the hyper parameters of the algorithms. In order to increase the precision value further, the threshold for the normal data percentages could be raised above 99.5% and 99.9% at the cost of lowering the recall value. The results tell us that with simple rules on how to define a threshold, an algorithm can find never seen before anomalies. Because of the simplicity of the data, it is hard to tell how the presented scheme scales to a more advanced network. It is also unknown how well the algorithms would perform in a network with larger variations in traffic congestion over time.

6.2 The drawbacks of synthetic network traffic in data generation

The choice to generate synthetic traffic was essential to the projects' time frame. It allowed us to focus on how to process the data and perform tests on the chosen machine learning algorithms. The drawback of this type of data is that it only allows the algorithms to analyze network flows, which means that the anomalies generated must differentiate from normal network traffic flows. This limited the set of anomalies that could be used to check the accuracy of the machine learning algorithms. Since the synthetic network traffic contains a dummy payload no deep packet inspection can be performed. No information can be gathered from the service selected either as all traffic is made to connect using the same service, even if they are on different ports. These are constraints that can be solved with a network setup that simulates human traffic. Using synthetic traffic allows the project to be more generalized, as the traffic generation is abstract and can be replaced with any type of connection and still be used for data gathering. The small size of the lab allowed us to map the traffic accurately creating data that details the specifics of the network. The low number of services running on the network is also a contributing factor to creating this detailed data. A larger network would be more complex which would mean that creating time bins this way would be unfeasible as the number of time bin features scale poorly. The features of the time bins are made in a way that a slight change in the network topology would require the algorithms to be re-trained.

6.3 The KDD '99 data set compared to the time series pre-processing

The KDD '99 data set contains 41 features which is a lot more than the 8 features extracted from the IP headers and more than the 17 features in the time bins. Many features of the KDD '99 data set are often dropped in machine learning benchmarks in order to speed up the training of the algorithms. It has been shown that the 41 features of the KDD '99 data set can be reduced to 8 features and still perform

well for anomaly detection [36]. The time series proved efficient in reducing the amount of collected network data and the results show that this is an effective way of data processing for discovering network anomalies.

It is hard to prove whether using time series is better or worse than using the other data processing methods mentioned in Section 3.4. Especially interesting is comparing this method to the connection-based technique since this is the type of pre-processing used in the KDD '99 data set. The time series created have very specific bins, containing a mapping of the entire network during a single second. This allows the machine learning algorithms learn the features of the network in its normal state. When creating a data set using connections from an already collected data set of packet data the process is trivial, as the data can be assumed to be ordered correctly. All connections are already complete, there are no mismatching packets and no connections are in progress. In a live network these problems are present and needs to be taken care of. Connections must also be classified as complete or incomplete and mismatched packets must be taken care of somehow. The result would be data that allows the machine learning algorithms to map network flows and still be able to check packet payloads. In the synthetic network data the payloads are irrelevant which makes time series an excellent pre-processing choice, in a real network however the payloads contain valuable information that can be used to detect more anomalies.

6.4 The labeling process of the anomalies

The labeling process of writing down the attack start and end did not work entirely well as the network could still be recovering from an anomalous state after the anomaly generating node has written an end. This would mean that the network state would be labeled as normal when it would still in fact be anomalous, a graph showing this can be seen in Figure 6.1. This seemed to only be an occasional problem with the node failure, which makes the accuracy a bit harder to detect. The SYN flood and the port scan are always labeled correctly.

The anomaly detection also performed better when the anomalies had longer time intervals between them. This is because of the small number of anomalies in the project, as the risk of getting a node failure following after a node failure is increased. A node failure following a node failure with a small amount of time between them causes the network to not recover between the node failures, making the time between them to be labeled as normal while it is actually anomalous. This causes some of the machine learning algorithms to perform worse.

It would be preferable to have several nodes performing the different anomalies. This would however create synchronization problems with the labeling. The limited time frame of this project did not allow us to create this type of anomaly generation. However all anomalies except the node failure are configured to hide the fact that only one node is performing them. The node failure would cause different decreases in network traffic depending on which node it is affecting since different nodes handle different amounts of traffic.

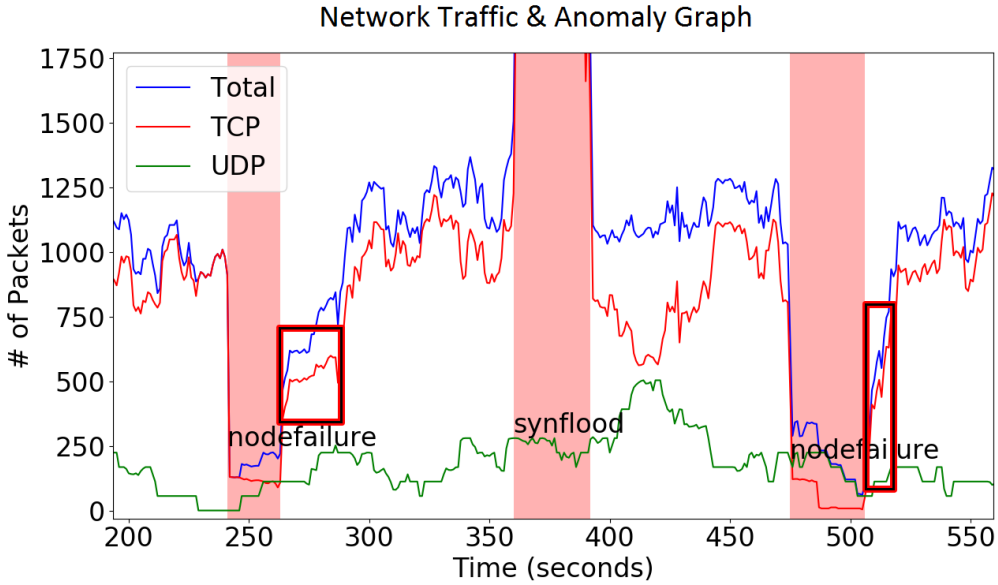


Figure 6.1: This graph shows how labeling the node failures can be inaccurate as the network is still in an anomalous state even after the node failure is over. The black and red squares show the regions where the network is labeled as normal while still in an anomalous state.

6.5 Setting up rules and thresholds for the machine learning algorithms

The three machine learning algorithms were chosen mainly because they are unsupervised or self-supervised methods. Instead of learning specific network anomalies with supervised learning methods, the main goal of the algorithms were to set up rules for what is normal network traffic. If new network traffic data deviates from the rule it will be labeled as anomalous. Another reason why the specific algorithms were chosen was because they set up different rules in order to detect outliers when looking at the network traffic data. The One-class SVM methods tries to find the optimal hypersphere that encloses the normal network traffic data points. By implementing these boundaries into a live detection network the network traffic data is labeled anomalous or normal depending on if the data point is inside or outside of the boundary. In the second method when using a LSTM network, the idea is to learn the network to predict the next state of the network (i.e. predicting the features' values in the next second). Then by taking the root mean squared error of the prediction and the ground truth classify the network traffic data as anomalous or normal depending on the magnitude of the error. The last method is the k-means clustering algorithm, which defines cluster centers in high density regions of the normal network traffic data. When new network traffic data are presented, it is classified as anomalous or normal depending on the squared Euclidean distance from the closest cluster center.

The first problem encountered was to define methods to chose hyper parameters and thresholds for the different methods. The staring point is that the data available is unlabeled and the majority of the data is normal network traffic. The hyper parameters and the thresholds should be defined such that they identify the majority of the normal network traffic as normal. When looking at real network traffic it is hard to know if the actual network traffic is normal or if it contains an significant amount malicious traffic, therefore setting up a framework to include or exclude a part of the training data is preferable. For the One-class SVM this was done by training the algorithm while decreasing the two hyperparameters ν and γ as in Figure 4.7, the hyper

parameter values was chosen when the algorithm reached a predefined accuracy of the normal traffic data. The LSTM was trained on the normal traffic data set and 20% of the set data was used for validation to not overfit. The root mean squared error was calculated for each prediction and ground truth, see Figure 4.9, and with incrementing threshold values the overall accuracy was calculated. When the accuracy reached a predefined limit the threshold value was saved. The threshold of the k-means clustering algorithm was defined similar to the LSTM algorithm, though in this case the squared Euclidean distance, see Figure 4.5, was incremented and the accuracy calculated. Since the data is synthetic it was assumed that almost no outliers would be in the data training set. When training the algorithms on the gathered data the threshold was calculated such that 99.9 % of the traning data was labeled as normal for the One-class SVM and 99.5% for the LSTM and k-means algorithm.

One-class support vector machine

In the One-class SVM two questions arose. The first being which kernel function to use and the second which hyper parameter to choose. Since the Radial Basis Function is one of the most used kernels, this one was chosen. The values of the hyper parameters were chosen from the method described in Section 4.4.2. When comparing the results to reported results from papers using KDD '99 [45] it was noticed that the One-class SVM had a superior precision score compared to the one found in this report. The precision score shows how well the algorithm will not label a normal data point as anomalous. We have shown that the live network contains data that has been classified as normal that should be anomalous, which is a factor to the relatively poor precision of our One-class SVM. The algorithm proposed in this thesis has a higher recall score, resulting in a total F1 score that beats the one presented using the KDD '99 data set.

LSTM neural network

The LSTM neural network contains a lot of parameters and architecture that can be tweaked in order to increase the performance. Some examples would be batch size, time lags, optimizers, learning rate etc. Out of the three algorithms the results show that the chosen LSTM

model does not perform as well as the others. This does not mean that the algorithm is ill suited for anomaly detection, but rather that the tuning of the hyper parameters are off. When comparing the results of the LSTM in this project to the ones found in the ones found using KDD '99 [23] there is a similar decrease in precision to the increase in recall causing only a small increase in the F1-score. We conclude that the LSTM algorithm would perform better given further evaluation.

k-means clustering

The specific problems with the k-means clustering were mainly two major points. The first problem was deciding how many clusters to use. Since the number of clusters is a free parameter, this was done by testing different number of clusters and evaluating the results as seen in Figure 4.6. There was a tendency for larger number of clusters to perform better, but after 40 clusters the differences were faint and therefore 40 cluster were chosen in the final configuration. The second problem was that the k-mean algorithm does not guarantee convergence to the global minima. In order to check that the result is reliable, 10 different runs with the same configuration were run, where a new threshold for each run was calculated and then the results were evaluated.

6.6 The ethics of network surveillance

Applying machine learning algorithms on a network to discover anomalies brings up some ethical questions. The algorithm is effectively surveying network data to find patterns that has been defined as anomalous and can be classed as a network surveillance system. These algorithms can be configured to detect other traffic as well, enabling these algorithms to be used in order to violate a persons privacy. It has been known for quite some time that such systems are already in effect across the world [11]. This is not a problem with the machine learning algorithms themselves, but rather with how they are used. We will not argue if these systems are good or bad by themselves, however we wish to state that it is imperative that these types of network surveillance systems are developed keeping personal privacy in mind and that the data collected is kept secure. The methods de-

scribed in this project can be used to track private network data and the attacks performed can be used with malicious intent. There are no novel attacks described in this project and more information of the attacks can be found using various online resources. This led us to conclude that it is safe to have these attacks described and used in this project.

7
Conclusion

These are our final thoughts on the project as a whole, what machine learning algorithm performed best and suggestions on future works that can be done in the field. The results were similar to the ones found in reports using the KDD '99 data set. This proves that the methods explained in this thesis are able to produce an anomaly detection system with an accuracy similar to the ones developed using the KDD '99 data set without having the same drawbacks as when using the KDD '99 data set. The scores presented in Chapter 5 and the reasoning in Chapter 6 show that the k-means clustering algorithm and the One-class SVM produce similar results of success in anomaly detection while the LSTM needs further evaluation.

Future work

The network traffic in this project was gathered on a small scale computer network. A suggestion for a future project is to attempt the same methodology on a larger network. The data in this project is based on only synthetic computer network traffic. A step further would be to either gather real network traffic data or set up a lab that realistically simulates human traffic using real services. Having payloads that contain real network data allows for more advanced anomalies and more sophisticated pre-processing techniques. A comparison between different pre-processing techniques for live anomaly detection in computer networks is needed as well. These techniques should be more general than the ones proposed in this thesis. During the project we discussed whether it was possible to have a distributed anomaly detection system running on each of the hosts. The idea was never brought further due to the project's time frame but it would be possible to implement in the future. There are a lot of different and advanced machine learning algorithms used in anomaly detection and since network traffic data is

seldom labeled there is a need of a better framework on how employ these algorithms in a real world setting using unlabeled data.

Bibliography

[1] The association for computing machinery's special interest group on knowledge discovery and data mining. `https://www.kdd.org/kdd-cup/view/kdd-cup-1999/Tasks`. Accessed: 2019-04-05.

[2] D-ITG , distributed internet traffic generator. `http://traffic.comics.unina.it/software/ITG/`. Accessed 2019-03-15.

[3] iPerf, the ultimate speed test tool for TCP, UDP and SCTP. `https://iperf.f`. Accessed 2019-03-15.

[4] KDD Cup '99 dataset (Network Intrusion) considered harmful. `https://www.kdnuggets.com/news/2007/n18/4i.html`. Accessed: 2019-04-17.

[5] Nmap Security Scanner. `https://nmap.org`. Accessed 2019-03-15.

[6] S.A. Alexandropoulos, S. Kotsiantis, and M. Vrahatis. Data preprocessing in predictive data mining. *The Knowledge Engineering Review*, 34:e1, 2019.

[7] M. Alkasassbeh, G. Al-Naymat, and E. Hawari. Towards generating realistic snmp-mib dataset for network anomaly detection. *International Journal of Computer Science and Information Security ISSN 1947 5500*, Vol. 14:(pp. 1162–1185), September 2016.

[8] S. Avallone, S. Guadagno, D. Emma, A. Pescape, and G. Ventre. D-ITG Distributed Internet Traffic Generator. In *First International Conference on the Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings.*, pages 316–317, September 2004.

[9] S. Benferhat, K. Sedki, and K. Tabia. Preprocessing rough network data for intrusion detection purposes. July 2007.

[10] E. Bertino and N. Islam. Botnets and internet of things security. *Computer*, 50(2):76–79, February 2017.

[11] J. Betts and S. Sezer. Ethics and privacy in national security and critical infrastructure protection. In *2014 IEEE International*

Symposium on Ethics in Science, Technology and Engineering, pages 1–7, May 2014.

[12] E. Bou-Harb, M. Debbabi, and C. Assi. Cyber scanning: A comprehensive survey. volume 16, pages 1496–1519. Institute of Electrical and Electronics Engineers, March 2014.

[13] V. L. Cao, V. T. Hoang, and Q. U. Nguyen. A scheme for building a dataset for intrusion detection systems. In *2013 Third World Congress on Information and Communication Technologies (WICT 2013)*, pages 280–284, December 2013.

[14] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[15] H. Debar, M. Dacier, and A. Wespi. A revised taxonomy for intrusion-detection systems. volume 55, pages 361–378, July 2000.

[16] A. Divekar, M. Parekh, V. Savla, R. Mishra, and M. Shirole. Benchmarking datasets for anomaly-based network intrusion detection: KDD CUP 99 alternatives. In *2018 IEEE 3rd International Conference on Computing, Communication and Security (ICCCS)*, pages 1–8, October 2018.

[17] J. Flood, M. Denihan, A. Keane, and F. Mtenzi. Black hat training of white hat resources: The future of security is gaming. In *2012 International Conference for Internet Technology and Secured Transactions*, pages 488–491. Institute of Electrical and Electronics Engineers, December 2012.

[18] A. Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385. Springer-Verlag Berlin Heidelberg, January 2012.

[19] L. Han. Research of k-means algorithm based on information entropy in anomaly detection. In *2012 Fourth International Conference on Multimedia Information Networking and Security*, pages 71–74. Institute of Electrical and Electronics Engineers, November 2012.

[20] S. Hochreiter and J. Schmidhuber. Long short-term memory. volume 9, pages 1735–1780, Cambridge, MA, USA, 1997. MIT Press.

[21] S. Hsiao and D. Kao. The static analysis of wannacry ransomware. In *2018 20th International Conference on Advanced Communication Technology (ICACT)*, pages 153–158. Institute of Electrical and Electronics Engineers, February 2018.

[22] M. Jianliang, S. Haikun, and B. Ling. The application on intrusion detection based on k-means cluster algorithm. In *2009 International Forum on Information Technology and Applications*, volume 1, pages 150–152. Institute of Electrical and Electronics Engineers, May 2009.

[23] J. Kim, J. Kim, H. L. T. Thu, and H. Kim. Long short term memory recurrent neural network classifier for intrusion detection. In *2016 International Conference on Platform Technology and Service (PlatCon)*, pages 1–5. Institute of Electrical and Electronics Engineers, February 2016.

[24] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*. Ithaca, NY: arXiv.org, December 2014.

[25] D. Lee, B. E. Carpenter, and N. Brownlee. Observations of UDP to TCP ratio and port numbers. Institute of Electrical and Electronics Engineers, January 2010.

[26] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, pages 281–297, Berkeley, Calif., 1967. University of California Press.

[27] R. Madan and P. SarathiMangipudi. Predicting computer network traffic: A time series forecasting approach using dwt, arima and rnn. In *2018 Eleventh International Conference on Contemporary Computing (IC3)*, pages 1–5. Institute of Electrical and Electronics Engineers, August 2018.

[28] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. volume 5, pages 115–133. Bulletin of mathematical biophysics, 1943.

[29] M. Moore. Penetration testing and metasploit. April 2017.

[30] R. T. Rockafellar. Lagrange multipliers and optimality. volume 35, pages 183–238, Philadelphia, PA, USA, June 1993. Society for Industrial and Applied Mathematics.

[31] D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. volume 323, pages 533–536. Nature Research, October 1986.

[32] H. Sak, A. W. Senior, and F. Beaufays. Long short-term memory

based recurrent neural network architectures for large vocabulary speech recognition. volume abs/1402.1128. International Speech Communication Association, 2014.

[33] R. Samrin and D. Vasumathi. Review on anomaly based network intrusion detection system. In *2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*, pages 141–147. Institute of Electrical and Electronics Engineers, December 2017.

[34] C. L. Schuba, I. V. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, and D. Zamboni. Analysis of a denial of service attack on TCP. In *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No.97CB36097)*, pages 208–223. Institute of Electrical and Electronics Engineers, May 1997.

[35] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. Platt. Support vector method for novelty detection. volume 12, pages 582–588. MIT Press Cambridge, MA, USA, January 1999.

[36] L. Shilpa, J. Sini, and V. Bhupendra. Feature reduction using principal component analysis for effective anomaly–based intrusion detection on NSL-KDD. volume 2. MultiCraft, July 2010.

[37] R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE Symposium on Security and Privacy*, pages 305–316, May 2010.

[38] O. Spatscheck, K. Levchenko, C. Kreibich, and S. Savage. Machine learning based traffic classification using low level features and statistical analysis. The International Journal of Foundations of Computer Science, December 2014.

[39] D. C. St. Clair. Learning programs. volume 11, pages 19–22. Institute of Electrical and Electronics Engineers, October 1992.

[40] M. Sundermeyer, R. Schlüter, and H. Ney. LSTM Neural Networks for Language Modeling. Institute of Electrical and Electronics Engineers, September 2012.

[41] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the KDD CUP 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pages 1–6. Institute of Electrical and Electronics Engineers, July 2009.

[42] N. N. Thi, V. L. Cao, and N.A Le-Khac. One-class collective

anomaly detection based on long short-term memory recurrent neural networks. volume abs/1802.00324. Ithaca, NY: arXiv.org, 2018.

[43] A. Varet and N. Larrieu. How to generate realistic network traffic? In *2014 IEEE 38th Annual Computer Software and Applications Conference*, pages 299–304, July 2014.

[44] A. Warzyński and G. Kołaczek. Intrusion detection systems vulnerability on adversarial examples. In *2018 Innovations in Intelligent Systems and Applications (INISTA)*, pages 1–4. Institute of Electrical and Electronics Engineers, July 2018.

[45] M. Zhang, B. Xu, and J. Gong. An anomaly detection model based on one-class SVM to detect network intrusions. In *2015 11th International Conference on Mobile Ad-hoc and Sensor Networks (MSN)*, pages 102–107. Institute of Electrical and Electronics Engineers, December 2015.

A

Appendix 1

A.1 KDD cup 1999 data overview

Below is a listing showing the features of the KDD cup 1999 data set. The first line is the labels used for the different connections. The indentation is made to show a line continuation. The rest of the lines show the feature name and how the feature is represented. Symbolic means that the feature contains strings of text. Continuous means that the feature contains a float number.

kddcup.names:

```
back , buffer_overflow , ftp_write , guess_passwd , imap , ipsweep ,
    land , loadmodule , multihop , neptune , nmap , normal , perl ,
    phf , pod , portsweep , rootkit , satan , smurf , spy , teardrop ,
    warezclient , warezmaster .
duration: continuous .
protocol_type: symbolic .
service: symbolic .
flag: symbolic .
src_bytes: continuous .
dst_bytes: continuous .
land: symbolic .
wrong_fragment: continuous .
urgent: continuous .
hot: continuous .
num_failed_logins: continuous .
logged_in: symbolic .
num_compromised: continuous .
root_shell: continuous .
su_attempted: continuous .
```

```
num_root: continuous.
num_file_creations: continuous.
num_shells: continuous.
num_access_files: continuous.
num_outbound_cmds: continuous.
is_host_login: symbolic.
is_guest_login: symbolic.
count: continuous.
srv_count: continuous.
serror_rate: continuous.
srv_serror_rate: continuous.
rerror_rate: continuous.
srv_rerror_rate: continuous.
same_srv_rate: continuous.
diff_srv_rate: continuous.
srv_diff_host_rate: continuous.
dst_host_count: continuous.
dst_host_srv_count: continuous.
dst_host_same_srv_rate: continuous.
dst_host_diff_srv_rate: continuous.
dst_host_same_src_port_rate: continuous.
dst_host_srv_diff_host_rate: continuous.
dst_host_serror_rate: continuous.
dst_host_srv_serror_rate: continuous.
dst_host_rerror_rate: continuous.
dst_host_srv_rerror_rate: continuous.
```