UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR IT-SICHERHEIT

Anomaly Detection for Intrusion Detection

Anomalieerkennung für Angriffserkennung

Masterarbeit

im Rahmen des Studiengangs
Informatik
der Universität zu Lübeck

vorgelegt von
Lucas Bergmann

ausgegeben und betreut von
Prof. Dr. Thomas Eisenbarth

mit Unterstützung von Claudius Pott und Dr. Simon Walz

Lübeck, den 7. Dezember 2020

Abstract

Today almost all digital devices have an active access to the internet. This makes them a reachable target for attackers. The detection of attacks on hardware devices are made by discovering revealing information in network packets. Furthermore, already infected hardware devices will produce revealing information in their network traffic. The data packets that are caused by the attacker differs from the standard behavior of benign data. Thus anomalies in the network traffic indicates that an attack is taken place. This master thesis deals with the intrusion detection in industrial networks. It shows which preprocessing steps are useful for a good classification. Furthermore, this thesis shows that is possible to train a working classifier without anomalies in the training set. Only a data history of benign network traffic is sufficient. This thesis also shows that network traffic captured in a real industrial facility has a lower complexity than the average network traffic. Thus, anomalies are easier to detect.

Heutzutage sind fast alle digitalen Geräte mit dem Internet verbunden. Dies macht diese zu erreichbaren Zielen von Angreifern. Das Erkennen der Angriffe auf die Hardware erfolgt durch das Erkennen von verräterischen Informationen in Netzwerkpaketen. Bereits infizierte Hardware produziert in ihrem Netzwerkverkehr verräterische Informationen. Datenpakete, die von einem Angreifer stammen, unterscheiden sich vom Standardverhalten von harmlosen Netzwerkverkehr. Deshalb deuten Anomalien im Netzwerkverkehr auf einen akuten Angriff hin. Diese Masterarbeit behandelt die Angriffserkennung in industriellen Netzwerken. Sie zeigt, welche Vorverarbeitungsschritte sinnvoll sind für eine gute Klassifizierung. Des weiteren zeigt dies Masterarbeit, das es möglich ist, ohne Anomalien im Trainingsdatensatz einen funktionierenden Klassifikator zu trainieren. Nur einen Datenhistorie mit harmlosen Netzwerkverkehr ist notwendig. Diese Masterarbeit zeigt außerdem, dass Netzwerkverkehr aus Industrieanlagen weniger komplex ist als durchschnittlicher Netzwerkverkehr. Dies erleichtert das erkennen von Anomalien.

Erklärung

Ich versichere an Eides statt, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

Lübeck, den 7. Dezember 2020

Contents

1 Introduction

Today almost all economic processes are digitized. This development was driven by the increase in productivity that could be achieved by introducing digital processes. Until now the systematic evaluation of the information contained in digitally recorded data has been neglected. Competitive advantages remain unused, which might threaten future business models.

The reasons for poor data analysis are manifold and may vary from sector to sector. This thesis deals with data analysis in the manufacturing industry. Frequent investment and adjustment of the production process impedes a widespread analysis of production data in this area. However, depending on the real circumstances and situation of the production process, this widespread analysis can be worth it.

In the manufacturing industry the cycle of innovation is comparatively long. Machines and software have to be amortized until they are replaced by a new one. Different depreciation periods lead to a high variance of production machines. These production machines have to be embedded in a data logging and analyzing context. Another cost driver is the training needed by the employees when switching to new technical equipment. An additional challenge is the risk minimization of an outage when introducing new technical equipment to the production facilities.

Old systems can also affect the security of the whole network. For example, a machine control system running on Windows XP does not present a problem if it is a stand alone system. However, if the system is connected to the internet over a network, it is a huge security risk for the whole production facility, because Microsoft dropped the official support[1] for Windows XP in 2014. An attacker can use a known and sometimes publicly available exploits. For the attacker, this makes it easy to infiltrate a production facility and to cause an economic loss to the attacked company. Besides, an intentionally manipulated machinery can be a safety risk to all employees working with it. If a company decides, due to financial aspects, to run an outdated machine control system, it has to be ensured that an attacker does not have the opportunity to run an exploit on the target system.

[1] Microsoft offered a longer custom support for individual solutions.

1.1 Intrusion Detection

The absence of a connection to the internet is one solution to prevent a system from being infected with malware. Most times, this is not a practical solution. A comprehensive analysis needs multiple data sources joined together and therefore all systems need to be connected through a network. An internet connection is also needed if the production facility is split up on multiple locations. The same applies to the remote monitoring and maintenance of facilities. If a production machinery is only connected to a central computer with no internet connection, the network can be compromised e.g. by an infected USB flash drive. Instead of hoping that the employees are following the companies security rules, it makes sense to take technical precautions in order to prevent an infection because employees might not remember all security rules or breaking them on purpose or for a more efficient workflow. Furthermore, an external person, whose presence might be authorized, can infect a computer or machinery by purpose.

Comparable challenges do also exist in other economic sectors. All medical devices in hospitals or a doctor's office have to be connected securely. This is the only way to prevent an extraction of critical patient data or an manipulation of the medical devices. An unprotected access to a building control system and its underlying infrastructure with its sensors and actuators is a huge security flaw.

A firewall only gives a rudimentary protection against some attacks, but it is not appropriate for a full protection or a separation of a network. A firewall does only follow simple predetermined security rules. It can only block or allow IP addresses or ports for programs, but an authorization technique is usually not supported.

In this regard the Security Gateway developed by cbb software GmbH finds a remedy. It provides multiple APIs which allow remote monitoring and maintenance of a machinery. A direct connection to a machinery is prohibited by the gateway. The machinery is physically separated from the facility network by the Security Gateway. Therefore, an update of the machinery software for security fixes is not necessary anymore. In fact, the Security Gateway is the only item that needs an update if a new security threat is established. If a malfunction occurs after an update of the Security Gateway, the old functional software can be easily restored.

The Security Gateway developed by cbb software GmbH provides a high security level against malicious attacks. A better protection is a strong selling point in the security

software market. For this reason an intelligent differentiation of malicious and benign network traffic is desirable. A system that detects malicious network requests and takes eventually counteractions automatically increases the security even more. Furthermore, an infected system can be detected in time for an administrator to isolate this system before it can take action. This thesis examines if either classical algorithms or machine learning are the best basis to accomplish this tasks.

1.2 Analyses of Production Data

An artificial intelligence which detects anomalies in network data can also be used to detect anomalies in sensor data of a production facility or machinery. The data of the sensors reveals more about the production process or a single machinery than is apparent at first glance. An artificial intelligence is able uncover causalities within the huge amount of sensor data. It identifies the most significant parameter for the quality of a product. This way, new options for quality control techniques can be established. Besides, the ratio of products that do not meet the required quality standard can be reduced. This method increases the effective output of a production and thus reduces the cost per unit. The production facility can work more efficient.

Furthermore, this method can increase the estimation quality of the prediction of a technical failure risk. Most of the machines in a production facility have multiple sensors to monitor their current state. By a data history of these machines the current measurements will be compared with old ones. If a derivation is detected, this might indicate an advanced attrition of a machine, which will lead to a failure in the near future. A software system that is following these steps provokes an early maintenance of an affected machinery. This way, the wearing part might be identified and replaced in time before a total malfunction occurs. Disturbances in operation are prevented this way as well. In summary, anomaly detection in production data can support the whole production, it can increase the quality of the final product and can reduce the downtime of a machine.

1.3 What are Anomalies?

Anomalies are patterns in data which differ from patterns observed in normal data. They are generated by abnormal activity, e.g. malicious activity. There are three types of anomalies:

Point anomaly When a single data point differs in a strong way from the rest of the points, this single point is a point anomaly. This means that regular points form

one or multiple clusters and the single point is not located in or near such a cluster. For example an unusual high debit to a bank account can represent an indication for a credit card fraud.

Contextual anomaly When a data point differs from the rest of the points within the same context, it is a contextual anomaly. The attributes of points are separated in contextual and behavioral attributes. Contextual attributes represent the context or neighborhood, behavioral attributes describe non-contextual attributes. For example a measured temperature of $0°C$ is not unusual in January, but it is a anomaly for a month like July. Also the location of a weather station has an effect to the context and therefore determines if a point is a anomaly. In this example longitude, latitude and time of the year are contextual attributes, the measured temperature is a behavioral attribute.

Collective Anomaly When a group of data points together behave anomalously, this is a collective anomaly. A point of this collective anomaly does not have to be anomalous. For example in an UDP data stream it can occur that a few packets get lost but if almost all packets cannot reach the destination then there is an anomaly.

1.3.1 Challenges

It is not easy to detect anomalies. One reason for this is that there is no precise definition of normality for abstract circumstances. Chandola et al. highlight 6 major challenges for anomaly detection[CBK09]:

- *Defining a normal region which encompasses every possible normal behavior is very difficult. In addition, the boundary between normal and anomalous behavior is often not precise. Thus an anomalous observation which lies close to the boundary can actually be normal, and vice-versa.*
- *When anomalies are the result of malicious actions, the malicious adversaries often adapt themselves to make the anomalous observations appear like normal, thereby making the task of defining normal behavior more difficult.*
- *In many domains normal behavior keeps evolving and a current notion of normal behavior might not be sufficiently representative in the future.*
- *The exact notion of an anomaly is different for different application domains. For example, in the medical domain a small deviation from normal (e.g., fluctuations in body temperature) might be an anomaly, while similar deviation in the stock market domain (e.g., fluctuations in the value of a stock) might be considered as*

normal. Thus applying a technique developed in one domain to another is not straightforward.

- *Availability of labeled data for training/validation of models used by anomaly detection techniques is usually a major issue.*
- *Often the data contains noise which tends to be similar to the actual anomalies and hence is difficult to distinguish and remove.*

1.3.2 Input and Output for Anomaly Detection

There are two possible options to report an anomaly when it is detected. With a scoring technique, a score is assigned for each instance. Later, a threshold determines if this instance represents an anomaly or not. This has the advantage to move the detection bias towards false positive or to false negative. The labeling technique only makes a difference between normal and anomalous instances. This technique is more limited for different requirements from various applications but as an advantage it may have a faster computation time.

Attributes of input data can be categorized in three groups. These are binary, categorical, and continuous. A binary attribute can only have two states which are either True or False, or One and Zero, respectively. Categorical attributes have a finite (but more than two) non numerical set of possible values, for example the name of the origin service by a network packet. Continuous attributes are numerical and in theory infinite. Practically they are bounded by the numerical data type that is used. Each data instance can have one or more attributes, even with different types.

1.4 Types of Network Attacks

An attack on a network or a computer can have different aims. Therefore, the methods of attacking differ and for this reason the attacks can be differentiated into four groups:

DoS Short form for denial of service. The aim of the attacker is to limit the functionality of the network or to make it unusable. It can also be used to disguise other attacks.

Probe This is used to gain some information about the network, computers and their provided services. At first glance this attack does no damage and seems harmless. But the gathered information can be used to find usable security flaws for further and harmful attacks.

User to Root Short form U2R, an attacker aims to get more privileges (mainly root access) than his current role is supposed to have.

Remote to User Short form R2U, an attacker aims to get local access to a system from remote. If he gets local access he can send network packets in the local network for further attacks or use U2R attacks to raise his privilege level.

1.5 Security Gateway

The Security Gateway is a hardware/software solution to secure a subnetwork from the outside. It is switched between network and subnetwork. Therefore, every network packet has to pass the Security Gateway to reach the subnetwork. Every network packet is transmitted from the network to the subnetwork or from the subnetwork to the network, respectively. The middleware translates the network packet from one software interface to another. In Figure 1.1 the exact structure of the Security Gateway is displayed. Network packets that are to be transmitted from the insecure network (*Red Network*) to the secured subnetwork (*Green Network*) are routed to the right software interface which is determined by the port number. The software interface handles the packet and sends a network packet with the information to the *Green Network* if it is conform with the specification.

Due to this structure all machines in the subnetwork are not visible from the network and only specific data reaches the subnetwork and its connected machines. Consequently, the Security Gateway is the perfect place to analyze network traffic and to find anomalies in these.

1.6 Outlook

Based on the problem described above, intelligent algorithms are designed which can detect anomalies in data. Each algorithm is designed to fulfill most of the following requirements sufficient as possible:

Point anomaly The algorithm shall detect anomalous data points. In the described use cases contextual or collective anomalies do not take place.

Scores If an algorithm produces too many false positives it is an unnecessary stress for a system admin. Besides, after some false positives it might happen that a system administrator does no longer take alerts seriously. So an anomaly can be overseen and can be harmful for the system. With a scoring system the false positive rate

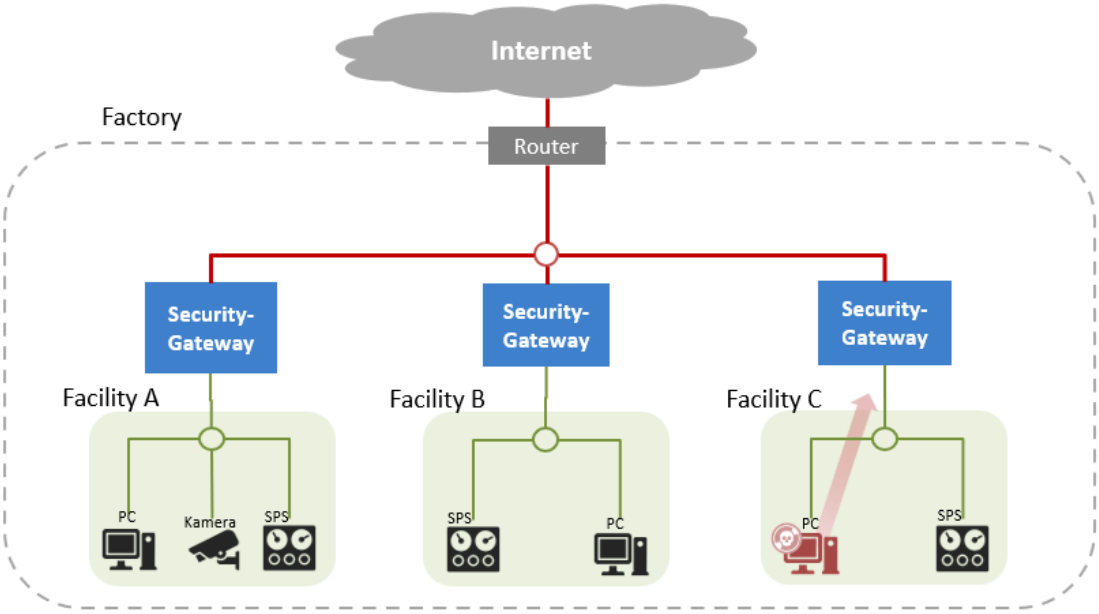

Figure 1.1: Structure of the Security Gateway. The Security Gateway divides the network in a *Red Network* and a *Green Network*. On top in red there is the potential insecure *Red Network* which is connected to the internet. On the bottom are three *Green Networks* which are secured by Security Gateways. Thus, an infected PC, even if it is in a *Green Network*, does not reach other devices in other *Green Networks*

can be reduced by raising the classifiers threshold after detecting a high rate of false positives.

Performance Fast classification and low memory consumption is an important aspect. Slow classification can cause packet drops in a network or it decreases the detection rate by skipping data instances. Furthermore, a faster classifier allows less expensive hardware.

Identify relevant feature Data instances consist of many features. But sometimes only a few features of an instance can be sufficient to reveal an anomaly. In order to discover which part of a machine is worn out, it can help, for instance to identify the feature that causes the classifier to detect an anomaly.

Two different main approaches for detecting anomalies are pursued, namely supervised and unsupervised learning. This master thesis mainly focuses on unsupervised learning, because labeling data is a huge effort and may be necessary for each intended use.

The evaluation of the classifier is mainly done on network data. Therefore, only gen-

eral information of the packets are used. These are for example header information or the amount of data is transferred. Deep packet inspection is only considered separated and shortly.

Identifying relevant features is the least important requirement because it is not necessary for network intrusion detection; it is only relevant for the analysis of production data. Therefore, classifiers are not evaluated on their ability to show these relevant features, only a assessment if they support this requirement takes place.

All in all, all approaches are based on classical algorithms or machine learning, for example autoencoder, k-distance, PCA and neural networks.

2 Basics

Both, anomaly detection and intrusion detection have been examined for a long time. So there are already a bunch of basics and fundamentals and are explained in the following paragraphs. Also fundamentals from other topics are used for this thesis. These are introduced in this chapter.

2.1 Datasets

A dataset is essential for the training of a classifier. It contains samples and mainly labels for each sample. The samples are divided into training set and a test set, sometimes also into a validation set. The reason for this is that the evaluation of a classifier on the training set can have a higher accuracy because the classifier recognizes the test data. The goal is to design a classifier that can generalize the behavior in the training set to classify new unseen instances right. This generalization is tested with a test set. Datasets are also used because they allow to compare different solutions. Thus, when comparing two classifiers trained with the same dataset, the quality of them are not depending on the used dataset.

There are only few publicly available datasets for intrusion detection or anomaly detection. In this thesis, the *NSL-KDD* [UNB], *CSE-CIC-IDS2018* [CSE] and the *UNSW-NB15* [MS15] dataset are evaluated. Additional, a self-created dataset is used.

2.1.1 NSL-KDD-Dataset

The *NSL-KDD* dataset [UNB] was published by the *Canadian Institute for Cybersecurity* from the University of New Brunswick in 2009. It is built on the *KDD99* which in turn is built on the *1998 DARPA Intrusion Detection Evaluation Program* dataset of the MIT Lincoln Labs[KDD].

Back then an U.S. Air Force LAN was simulated for nine weeks. During this time all network packets in this simulated network were recorded. Additional attacks were simulated and also recorded. 41 features were selected for each network packet. These features contain information about the TCP connection, content features within a connection suggested by domain knowledge and traffic features computed by using a two-second time window.

In total 38 attack variants were recorded, 14 of them are exclusive present in the test data. The latter are intended to validate if and with which coverage unknown attacks variants are detected. The attacks are grouped into *DOS*, *R2L*, *U2R* and *probing*.

The original dataset of the MIT Lincoln Labs has a lot of redundancy. Among other problems, training data and test data are not completely distinct. In addition, simple detectable attacks are overrepresented in the data set. These problems were only partially solved by the KDD99 dataset. Therefore, the *Canadian Institute for Cybersecurity* further modified the dataset.

The authors of the *NSL-KDD* dataset highlight the following advantages[TBLG]:

> *The NSL-KDD data set has the following advantages over the original KDD data set:*
>
> - *It does not include redundant records in the train set, so the classifiers will not be biased towards more frequent records.*
> - *There is no duplicate records in the proposed test sets; therefore, the performance of the learners are not biased by the methods which have better detection rates on the frequent records.*
> - *The number of selected records from each difficulty level group is inversely proportional to the percentage of records in the original KDD data set. As a result, the classification rates of distinct machine learning methods vary in a wider range, which makes it more efficient to have an accurate evaluation of different learning techniques.*
> - *The number of records in the train and test sets are reasonable, which makes it affordable to run the experiments on the complete set without the need to randomly select a small portion.*
>
>   *Consequently, evaluation results of different research works will be consistent and comparable.*

2.1.2 CSE-CIC-IDS2018 Dataset

The *CSE-CIC-IDS2018* dataset [CSE] is a collaborative project between the Communications Security Establishment (CSE) and the Canadian Institute for Cybersecurity (CIC). The purpose of this dataset is to evaluate an intrusion detection system. Basis of data is a network with more than 400 local machines and 30 servers. Different operating systems are running on the machines, for example Windows 7, Windows 10 and some Linux systems.

The dataset includes the captured network traffic and system logs of each machine, 80 features were extracted out of this. Its raw data is over 200 GB large.

The examined attacks are Bruteforce, DoS, Web, Infiltration, Botnet, and DDoS+PortScan attacks.

2.1.3 UNSW-NB15 Dataset

The *UNSW-NB15* dataset [MS15] was created by the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) in 2015. This dataset has nine types of attacks, namely, *Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode* and *Worms*. It has 49 features per instance.

More than 100 GB of raw pcap files were captured in the network of the ACCS. It consist out of real benign activities and synthetic attacks. These raw data were converted with the IXIA PerfectStorm tool to a training set containing 175,341 records and a test set containing 82,332 records.

2.2 State of Research

Quamar Nizay et al. investigated the impact of preprocessing the *NSL-KDD* dataset by using and training an autoencoder [JNSA15]. It was trained to learn an abbreviated representation of each instance of the train data. With this preprocessed data and labels, which are provided by the *NSL-KDD* dataset, a self thought learning classificator (STL) was trained. A softmax regression classifier (SMR) was trained to measure the impact of the preprocessing. Therefor no preprocessing took place for the SMR.

The preprocessing affected the quality of the classification, as can be seen in Figure 2.1. The overall accuracy of the STL classifier is 88.39%, whereas SMR only reaches an accuracy of 78.06%. SMR shows a better performance than STL in the precision metric. STL has a higher Recall and F1-Score than SMR. Thus compared to STL, SMR shows a lower false positive rate than STL, but this comes at the price of a lower detection rate. Therefore, Quamar Nizay et al. have shown that preprocessing data with a self taught autoencoder produces a classifier with a higher detection rate.

Chuanlong Lin et al. studied whether recurrent neural networks are suitable for classifying instances of the *NSL-KDD* dataset[CYJX17]. The authors preprocessed the data by

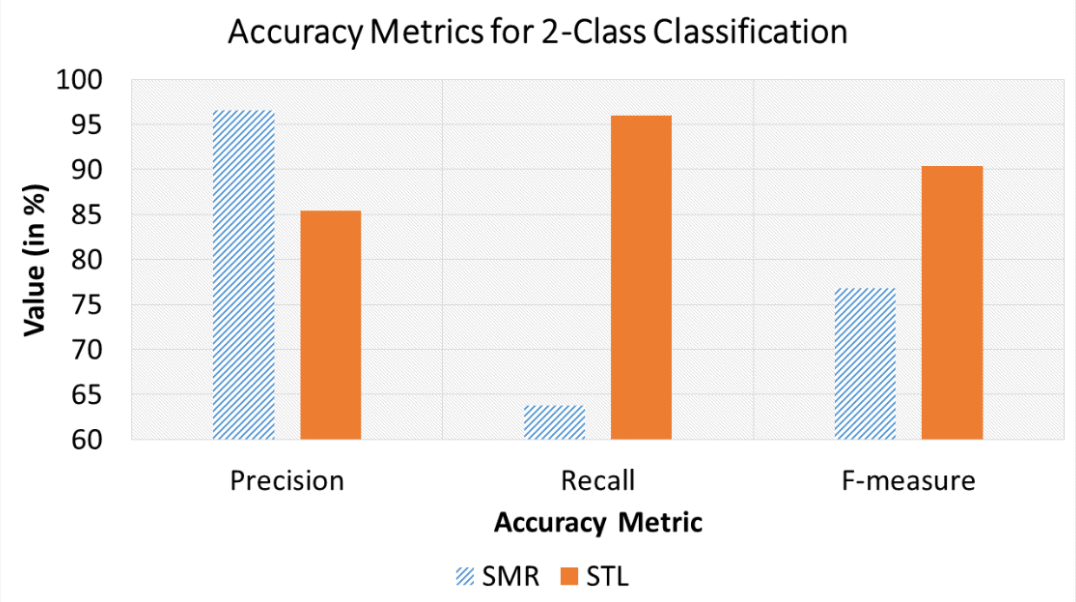

Figure 2.1: Results of self-taught-learning (STL) and softmax regression (SMR) by Quamar Nizay et al. This image is from the corresponding paper [JNSA15]. Preprocessing improves the recall of classifying, but decreases the precision. All in all, STL produces better results than SMR.

normalizing each feature independently. With these normalized data a recurrent network was trained. They have shown that their recurrent network achieved a higher accuracy than other methods. However, as it is shown in Figure 2.2, the recurrent network has an accuracy of $83,38\%$ and has thus only an insignificant advancement over other methods like Random Forest or Naive Bayes Trees.

Kejiang Ye et al. use a Long Short Term Memory (LSTM) to classify network data [LYX19]. A LSTM extends a Recurrent Neural Network with a forget gate, input gate and output gate. The authors use the CSE-CIC-IDS2018 dataset. They preprocessed the data by using SMOTE algorithm as data augmentation on underrepresented datasets. They also implemented an attention mechanism to extract and represent the relevant information.

The authors have shown that their approach leads to a high accuracy. In total they achieved an accuracy of 96% and a precision as well as a recall rate of 96% each. Also, the used data augmentation had a positive impact. Without any data augmentation web attack leads to a recall rate of 0.0% and infiltration attack to one of 11%. With data augmentation they gained a recall rate of 98% or 17%, respectively. Most attack types have a high recall and only few wrong classifications. Only the detection of web and infiltration

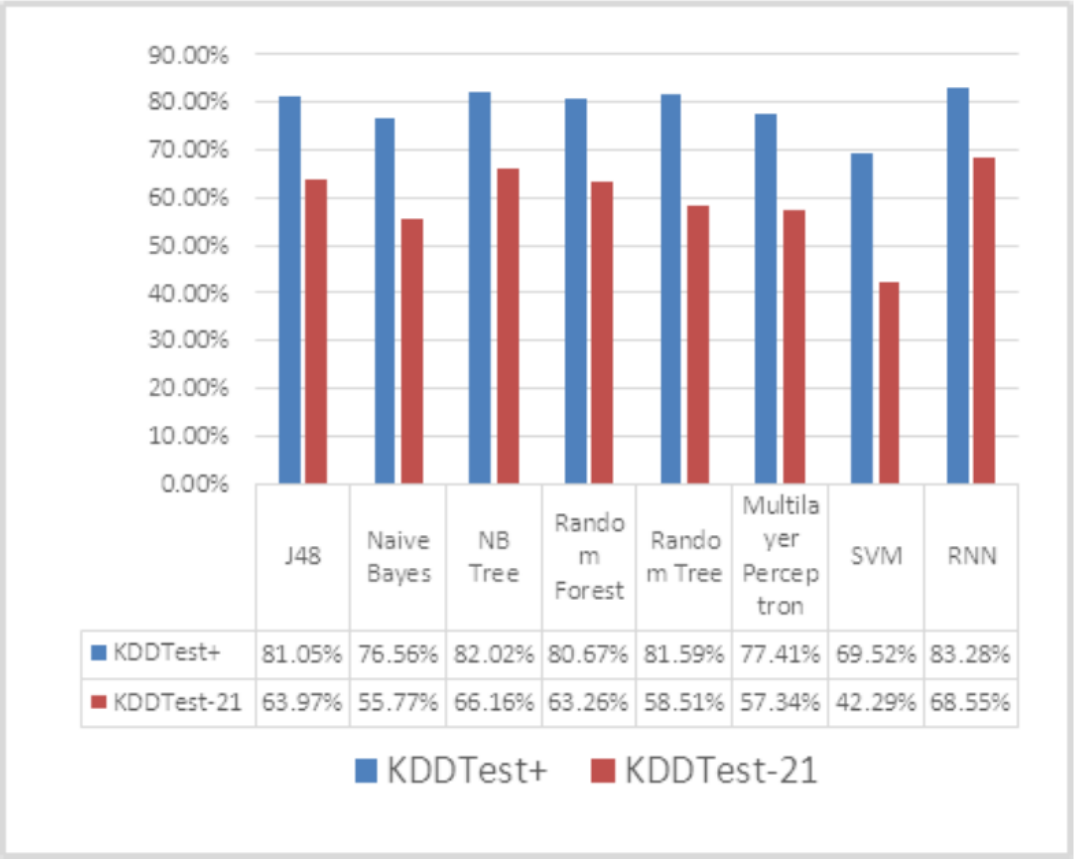

Figure 2.2: Results of the recurrent neural network (RNN) and results from other classification methods by Chuanlong Lin et al. This image is from the corresponding paper [CYJX17]. In the KDDTest-21 test dataset there are 21 various attack types and the goal is to distinguish these types.

attacks has a high failure rate. The authors argue that the underrepresentation of these attacks in the training set is the reason for these failures.

Nour Moustafa et al. used a beta mixture model (BMM) for detecting anomalies [MCS18]. The authors argued that, due to the elastic form of a beta distribution, features of network data cannot accurately fit a normal distribution. They evaluated the BMM on the UNSW-NB15 dataset. The BMM reached an accuracy of 93.4% and a recall of 92.7%.

Moustafa et al. also wrote a survey on network anomaly detection systems [MHS18]. The authors grouped the existing approaches into five categories, *Classification*, *Clustering*, *Knowledge*, *Combination* and *Statistics*. Their result was that each technique has advantages and disadvantages and that it depends on the circumstances if a technique is usable. For

example a combination technique has a long computation time but comes with a high accuracy and detection rates. Thus, this technique can be established in network regions with low traffic, but not on high frequented nodes. Additionally, they summarized the results of different approaches. However, different datasets were used for these approaches which limits their comparability.

Wressnegger et al. focused on the data stream to detect anomalies in network traffic [WKR18]. They collected 210 GiB of data from 92,700 unique devices. Therefor they used n-grams and counted the appearance of each n-gram to get a feature vector that always has the same length for each network packet even if the the network packet size differs. Since the possible number of different n-grams is very high for a high n, the authors used a Count-Min Sketch for reducing memory consumption while counting the occurrences. They showed that the counting error of the Count-Min Sketch is minimal. With the help of the generated feature vector a classification was done. For evaluating they added attacks in the dataset. As an result they could detect $97,8\%$ of the attacks with at most 1 false positive out of 10000 network packets.

2.3 Standard Methods for Classification

In the following section standard methods for classification and methods for supporting classification are listed. These methods are widely used for different classification tasks.

2.3.1 Data Collection

The first step of a classification is to collect the data which the classifier shall classify. For this task the CICFlowMeter [LDGMG17][GLMG16] is used. It is an open source tool to convert network traffic into flows and is mainly written in Java. For this it uses more than 80 statistical network features such as duration, number of packets, number of bytes and length of packets[2]. The advantage of the CICFlowMeter is that it does not provide categorical features. The length of the feature vector is thus equal in every network environment.

2.3.2 Data Preprocessing

Data preprocessing can be an important step to achieve better results in classification. It removes undesired data points, normalizes features and stretches the relevant number range of a feature.

[2]https://www.unb.ca/cic/research/applications.html#CICFlowMeter

Data Augmentation

A limited set of data can cause an overfitting of a classifier. In case of overfitting, the classifier does not learn to distinguish the instances by their characteristic, but to recognize instances and their corresponding labels. Overfitting mostly causes a worse classification of non training data. With more data an overfitting becomes unlikely. Therefore, new data instances are added to the dataset by creating them based on the existing instances of the data set. At first sight this seems counterproductive to data reduction, a method were some points of the training set are removed, but it is not. First of all, only few algorithms have a prediction computation time that is dependent on size of the training data. Furthermore, data reduction can remove a bias to a dense cluster; data augmentation can add a bias to a sparse cluster or does not add a bias.

One simple method of data augmentation is to duplicate each instance. To the duplicates a random noise is added. The duplication can be repeated several times until a sufficient[3] number of new points were generated. This method does not add a bias but it blurs a cluster boundary.

2.3.3 Normalization

Data instances have multiple features and each feature has another characteristic. So some features that are continuous can be in a negative and positive range, some only in the positive one. Also the maximum value is different for each feature. This makes it difficult to train a classifier. A solution to this is to normalize each feature so that all values are in $[0, 1]$. This is done in few steps for each feature separately. Step one is to check if the minimum of a feature is negative. If it is negative, the absolute value of the minimum value is added to each instances corresponding feature. In step two each feature is divided by its maximum. Doing this with all continuous features results the desired value range.

The classifier has to store each minimum and maximum of each feature. When it comes to classifying real world data, this data has to be normalized in the same way as the training data. It might happen that in the real world data some attributes are lower than the stored minimum or higher than the stored maximum. This causes the real world data not necessarily to be in the range of $[0, 1]$. This is a good thing because it is a strong hint that this data point is an anomaly.

[3] The sufficient number of instances is not well defined, it is up to the user to choose it wisely.

Binary features did not need to be normalized because they already are. Categorical features are converted into binary features. This is done by converting every state of this finite set into a binary feature. For example the set $\{red, blue, green\}$ is converted into three binary features, into the first one if the value is red, into the second if it is $blue$ and into the third one if it is $green$. This leads to more features and it is possible that some combinations of the features represent an invalid state. But these invalid combinations will never occur if the normalization algorithm is implemented correctly. Some datasets have a timestamp or a source and destination ip address feature. In this thesis, these features are removed during the normalization step because it is very likely that these will result in an overfitting of a classifier.

All in all, normalization is a necessary step for preprocessing data. Without it, it is for some algorithms impossible to learn (e.g. neural networks) and some algorithms like K-Nearest-Neighbor are overchallenged to work with different ranges in different dimensions. Additionally, this normalization step does not reduce the amount of information. Furthermore, normalizing each feature has a really fast computation time. Therefore, normalization is always used as the first preprocessing step for each classifier.

2.3.4 Transformer

A transformers task (sometimes also called scaler) is to stretch relevant parts of a feature space and reduces the irrelevant parts. An example for this is the time between two data packets in a connection. For most packets this frequency is high and only few packets have a frequency longer than a minute or even an hour. So the feature space of high frequency has to be extended and of low frequency decreased for using the whole feature space efficient. It has to be estimated automatically where these relevant feature spaces exist.

For every feature all values are, due to normalization in the preprocessing, in a range from zero to one. But these features are all distributed differently. So each feature has to be scaled individually. Also, one needs to consider that some features are categorical or are binary features which cannot be scaled properly.

Min-Max Scaler The simplest form of a scaler is a min-max-Scaler. It transforms all values between a minimum and a maximum. This scaler is not useful for preprocessing because this is already done by the normalization step and another range than 0 to 1 is not useful.

Power Transformer A power transformer is a non linear scaling method using power

functions. In this thesis the Yeo–Johnson transformation[YJ00] is used as a power transformer. This is a further development of the Box-Cox transformation [BC] and additionally allows zero and negative values.

Quantile Transformer A quantile transformer is a non linear transformation. It uses quantiles to get a uniform or a Gaussian distribution. This method can destroy linear correlations between two instances.

2.3.5 Dimension Reduction

With a dimension reduction the amount of effective features is reduced. A reduced set of features improves the performance of a classifier and might prevent overfitting. There are different methods to do this. Each of them tries to remove unnecessary information to have a better focus on the important information to make better classifications.

Principal Component Analysis

Principal Component Analysis (PCA) is a statistical procedure to get a lower dimensional embedding of data. It produces the most expressive linear combinations. An example is shown in Figure 2.3. The resulting vectors of the PCA are called eigenvectors and each vector has a corresponding eigenvalue. A higher eigenvalue means that the eigenvector has a lower reconstruction error. Thus, the top n eigenvectors are used. There are non linear extensions, for example kernel-PCA which uses the kernel trick to get a better lower dimensional embedding.

Transforming new points into the embedding space is simple. This can be done with the top n eigenvectors with few arithmetic operations. Therefore, this approach represents a fast dimension reduction method which minimizes the embedding error. Also, the eigenvector allows to reverse the dimension reduction with a small reconstruction error. This can be used to identify the features that are responsible for the classification result.

Isomap

Isomap is a nonlinear procedure to get a lower dimensional embedding of data. It shows huge similarity to PCA, the difference is that Isomap uses geodesic distance instead of euclidean distance. Thus, the main difference is that Isomap is able to detect manifolds and unfolds them to a lower dimension. This difference can be seen in Figure 2.3. Isomap learned to represent the 2D manifold in a 2D space, PCA only learned a 2D representation of the manifold structure.

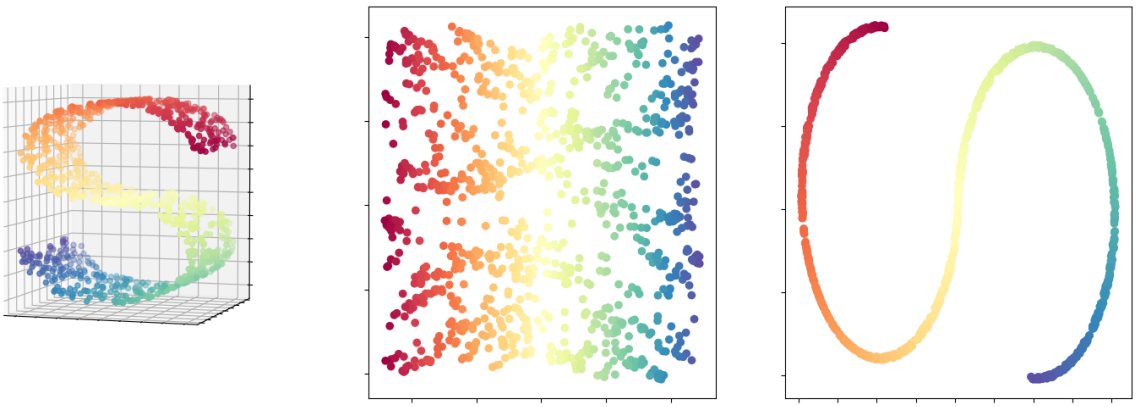

Figure 2.3: A manifold learning with 1000 points and a neighborhood size of 10. On the left hand side the original 3D data, in the middle a dimensional reduction with Isomap and on the right hand side a dimension reduction with PCA is displayed. This image is adapted from [skla].

For the Isomap algorithm a neighborhood size or an ϵ-range needs to be chosen. If its size is set to large the geodesic distance is not calculated correctly and its result converge to PCA result. If it is set too small some points might have an infinite distance to other points. In the worst case several non connected clusters are formed, all with infinity distance to each other. This would lead lead to a worse embedding.

A great disadvantage of Isomap is its runtime and memory consumption. The distance matrix of training points needs to be stored so the memory consumption scales quadratically with the number of points. Using 10.000 points and a measured distance accuracy of a 32 bit floating-point will result in a memory consumption of 3.2 GB, 1.000 points only in 32 MB. If a new point is supposed be transformed into embedding space, this point is added to the distance matrix and the eigenvalues and eigenvectors of this distance matrix are computed. With the eigenvectors and their corresponding eigenvalues the point can be embedded. This transformation is very runtime intense, but is to be determined if its embedding is much better than the other approaches.

Autoencoder

An autoencoder consists out of two neural networks, an encoder and a decoder. The encoder gets the original data point and translates it into a compressed representation. The decoder gets the compressed representation a tries to reconstruct the original input. It is trained by minimizing the loss between the original input and the reconstructed image.

In Figure 2.4 is a schema of an autoencoder is shown. The original image and the re-

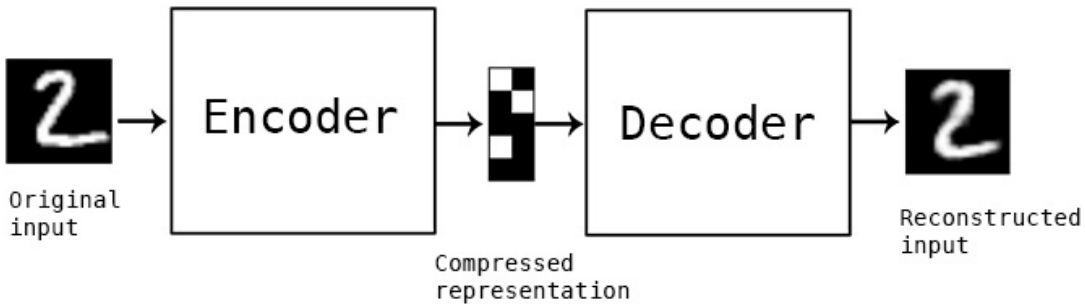

Figure 2.4: Schema[blo] of an autoencoder. In this example an gray scale input image of 28×28 pixel is used. The encoder compressed it to a size of 128 total pixel. This means a reduction of $83,7\%$ of the features took place. The decoder tries to reconstruct these 128 pixel to the original 28×28 gray scale image.

constructed image are similar, but there are also differences visible. However, in both images it is clear which digit is shown. So the encoder learned how to transform the input features into a lower embedding which the decoder learned to reconstruct.

This lower embedding of the input is the dimension reduction. It contains the most relevant information. An advantage of an autoencoder is that it does not need labeled data. If the encoder is a small neural network the processing time is low. On the other hand if in the embedded space an anomalous behavior of a feature has been detected, it is not clearly determinable which original features are responsible for this anomaly.

However, reverting the encoder neural network from the responsible output node of this anomalous feature can exclude some original features. This is done by multiplying the weights of the neural network backwards. If many original features are excluded so that only a few are left, it is possible to detect the responsible features of this anomaly. How many original features are excluded depends on the structure of the whole data and its distribution.

2.3.6 Classifier

A further step of anomaly detection is the classifier itself. It gets a (preprocessed) data instance and makes a decision which class this instance have.

k-Nearest-Neighbor

A classifying method that has been used since the beginning of machine learning is the k-Nearest-Neighbor algorithm. This algorithm uses lazy learning by only saving all training samples and by making the classification result dependent on this. With these data the algorithm does what the name says. It searches the k nearest neighbor in the training set of the to be classified instance and makes a majority vote of the label of the nearest neighbors. The wining label is the label the classifier classifies to this instance. The score of this classifier can be the percentage of the nearest neighbor having the same label.

Neural Networks

Nowadays, neural networks are widely used for different approaches and for many technical devices and are proofed to work well. With the appearance of APIs that are simple to use and well documented, creating and training an own neural network is simple. The difficulty lies in choosing the right structure and the right parameters in order to build a model of a neural network.

A neural network consists out of multiple layers. Each layer can be made out of a different type and the number of layers determines the deepness of the neural network. Each layer has an input vector and an output vector, the output of a layer is the input of the next layer. The values of the input and output vectors are always between zero and one.

The following layer types are the most commons:

Fully conncected The fully connected layer (sometimes called Dense layer) is the most universal layer. In this, each node of the layer is connected to all nodes in the previous layer. The number of nodes can be chosen freely. Due to the huge number of connections and thus a huge number of weights, too many nodes and fully connected layer can quickly lead to overfitting.

Convolutional The structure of convolutional layers correlates in some ways with the visual cortex of the human brain. It has some kernels each with the same chosen width and height. These kernels transform the input of the layer into a feature map. This layer has only few weights, which equals *number of kernels* $\times$ *width of kernel* $\times$ *height of kernel*. Therefore, it is not very so susceptible to overfitting. Also the input dimension and the output dimension are equal (or sometimes almost equal), which makes them useful for deep networks without overfitting.

On the downside, these kernels only have advantages for images or sound data and not for data from multiple sensor nodes of different features of network traffic. The reason is that on images neighboring pixels (features) are not independent from each other and if they differ too much from each other it is likely that an edge is detected. This way, it is easier to detect or recognize an object. On selected measured features this dependency between two features is not given. Therefore, convolutional layers are not used in this thesis.

Dropout The input dimension and the output dimension are always equal in a dropout layer. It has no weights, only a threshold between zero and one is chosen. Each value in the input vector that is smaller than this threshold is set to zero on the output. All values greater than the threshold are copied to the output. This layer shall reduce the probability of overfitting in the neural network.

Pooling A pooling layer is often used after a convolutional layer. There are three types of pooling layers. A minimum pooling, a maximum pooling and an average pooling layer. Mostly a pooling layer is a pooling with a size of 2×2. On this 2×2 array the minimum, maximum or average is written to the output. So a pooling layer has no weights and reduces the input dimension a lot.

Softmax A softmax layer is mostly used as the last layer in a neural network. Its input dimension is equal to its output dimension which is equal to the number of classes that are supposed to be distinguished. Its task is to do the final classification by returning an output vector whose sum of all values is one. This is mostly done with an Euler function. Also, this output shows how confident a neural network is with its classification. So a neural networks can score its prediction.

It is hard to determine why a neural network has classified an instance with a specific class. It shows the same problem as an autoencoder since an autoencoder are neural networks itself. The only possible solution is to multiply the weights backwards and to get an attention map of the neural network. With this it is possible to get the responsible features for a classification, but it depends on the structure of the data and the training set how the neural network learned it weights.

2.4 Reducing False Alarms

It is unlikely that a perfect classifier with no false alarms will be created in this thesis. Also, is it unlikely that an attacker attacks the network with a single data packet and can

manipulate the production with this single packet. Most attackers first have to analyze the network, prepare it and then execute the attack. This leads to several malicious network packets. That is why the system should only warn when there are several anomalous network packets.

However, saving the number of network packets classified as anomalous for each IP address is not practical. At least for UDP traffic an attacker can fudge the source ip address so that the list of anomalous ip addresses will grow to a not manageable size. Originally, an approach was developed for a different field of application that is suitable for this purpose, namely for the frequency estimation of internet packets with a limited space[DLOM02]. This algorithm works as follows:

1. Initialize T counter $< n, C >$ with $< 0, 0 >$, where n stands for the ip address and C is the counter value
2. For each positive classification for a given ip do:
   - If $< ip, C >$ with $C > 0$ exists, then increment C by one
   - Otherwise if $< x, 0 >$ exist, set it to $< ip, 1 >$
   - Otherwise decrement al counter by one

It is easy to see that this algorithm has a runtime and a space of $\mathcal{O}(1)$. After one ip address reaches a defined threshold, a warning is raised. This means that this happens only if an ip address has a significant higher amount of malicious classified network packets than other addresses. So the likeliness of false alarms is reduced with this simple algorithm.

3 Conception

3.1 New Methods for Classification

It seems that the listed standard methods are not the only ones capable of supporting a classifier. For example, data augmentation should consider the characteristic of the data and not only add a random noise. Also all classifier methods mentioned before need labeled data. Therefore, labeled data has to be already in or automatically added to the dataset or the classifiers must be adapted in such ways that they can detect anomalies without labeled training data. In this thesis, new methods have been designed for this purpose.

3.1.1 k-distance

The k-Nearest-Neighbor algorithm has a disadvantage when it comes to detecting anomalies. It only looks for the nearest neighbors but does not consider the distance between them. When all nearest neighbors are far away it is likely that the examined instance is an anomaly. Also, considering the distance, anomalies in the training data are not necessary. So the k-distance algorithm is used as an advancement of the k-Nearest-Neighbor algorithm.

The classifier measures the distance from the data instance i that is classified to the k nearest neighbor. If a benign neighbor is farther away than a specific threshold, this neighbor counts as an anomalous neighbor for i. A score for this neighbor is computed in relation to its distance. If this distance is short this data instance is very similar to a non anomalous data instance seen before. Therefore, this neighbor should count as benign with a low score. A majority vote determines the prediction, the score results from the score for each neighbor.

A huge advantage of this classifier is that it does not need labeled anomalies. Only historic benign data of the environment is needed but labeled anomalies might increase the detection rate. A disadvantage is that this classifier needs this historic data for each classification so it has to be hold in memory. The whole data has to be queried when this classifier has to do a classification. A k-d-Tree or a Ball-Tree can be used to query the data effectively but this takes $\log n$ time for each classification. Therefore, it is import to have

only few data points in the history but still enough to obtain a good classification.

One requirement for a classifier was that it returns a score for the classification. This score depends on the distance to the closest neighbor. A lower score means a lower probability of an anomaly. Also, an anomalous neighbor with a distance smaller than the threshold can increase this score, on the other hand multiple non anomalous neighbors can decrease this score.

With this approach it is simple to determine why an instance is labeled as an anomaly. The axis on which the distance to the next non anomalous instance is large are the features responsible for this anomalous classification.

Distance to Nearest Neighbor

First of all it is checked how close the instances are to their neighbors. From this, conclusions on the resemblance of data can be drawn. Different data bases can be used in order to do so.

The first data basis is the whole training dataset. For each instance of the training dataset the distance to the closest neighbor instance is computed. Here, four different cases can occur: First a benign instance has a benign neighbor (type 1), second an anomalous instance has a anomalous neighbor (type 2), third a anomalous instance has an benign neighbor (type 3) and fourth an anomalous instance has a benign neighbor (type 4).

In Figure 3.1 the distances in the training sets are visible. The x axis represents the euclidean distance to the closest neighbors, it is divided in 1000 quantiles and the nearest 100 are displayed. The size of each quantile is $1/1000$ of the maximum distance that has occurred. For the y axis the amount of instances with a distance to its neighbor matching the quantile is counted. In this top 100 quantiles lays for the *NSL-KDD* dataset 123505 out of 125956 instances, 97486 out of 100995 instances for *UNSW-NB15* and 821503 out of 823403 for *CSE-CIC-IDS2018* . This means that outliers are rare in all dataset and most instances are lying in clusters. Besides, it can be seen that most instances are of type 1 or type 2 and only a few are type 3 or type 4. In the *NSL-KDD* dataset 151 instances are of type 3 and 150 are of type 4, in the *UNSW-NB15* dataset are 5855 of type 3 and 5142 of type four. In the *CSE-CIC-IDS2018* only 6 instances are of type 3 and 9 of type 4. But it can be also seen that datasets behave differently from each other. The *UNSW-NB15* dataset seems to have clusters with various density and the higher rate of type 3 and type 4 instances indicates that some anomalous clusters and benign clusters are really close to

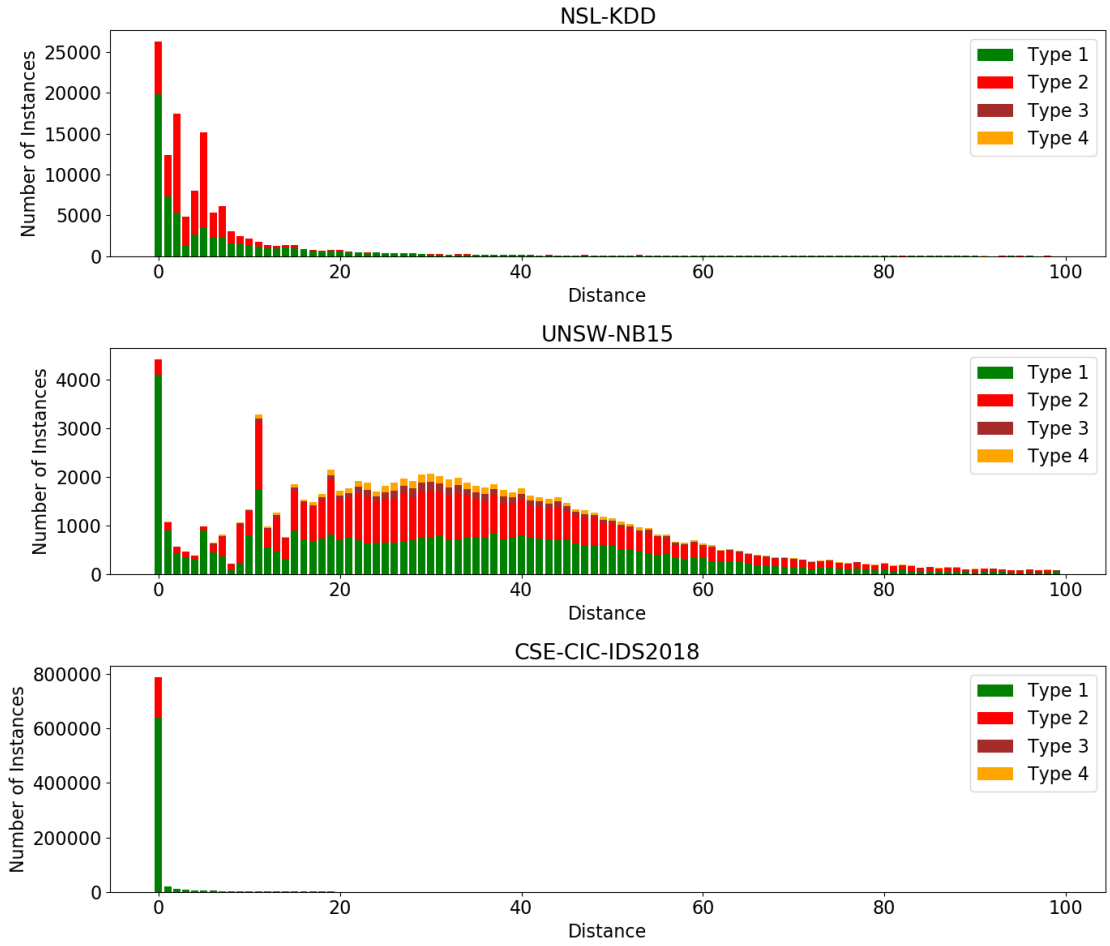

Figure 3.1: Distance of data instances to their neighbors. On top the *NSL-KDD* dataset, in the middle the *UNSW-NB15* and on the bottom the *CSE-CIC-IDS2018* dataset is shown. Green represents type 1, red type 2, orange type 3 and brown type 4.

each other. This can be seen by the higher amount of instances with an higher distance. In contrast, the *NSL-KDD* dataset consist of denser clusters and anomalous and normal clusters seem to be well separated. The *CSE-CIC-IDS2018* shows an extreme behavior. It has really dense clusters and well separated normal and anomalous clusters. This can be explained by the huge amount of instances in particular. The more instances exist, the more likely it is for each instance to have a very similar instance. This corresponds to the huge amount of duplicates in the dataset that needed to be filtered out. However, this cannot be the only explanation of this behavior. The test arrangement also has to have an impact on the sparse diversification of the instances.

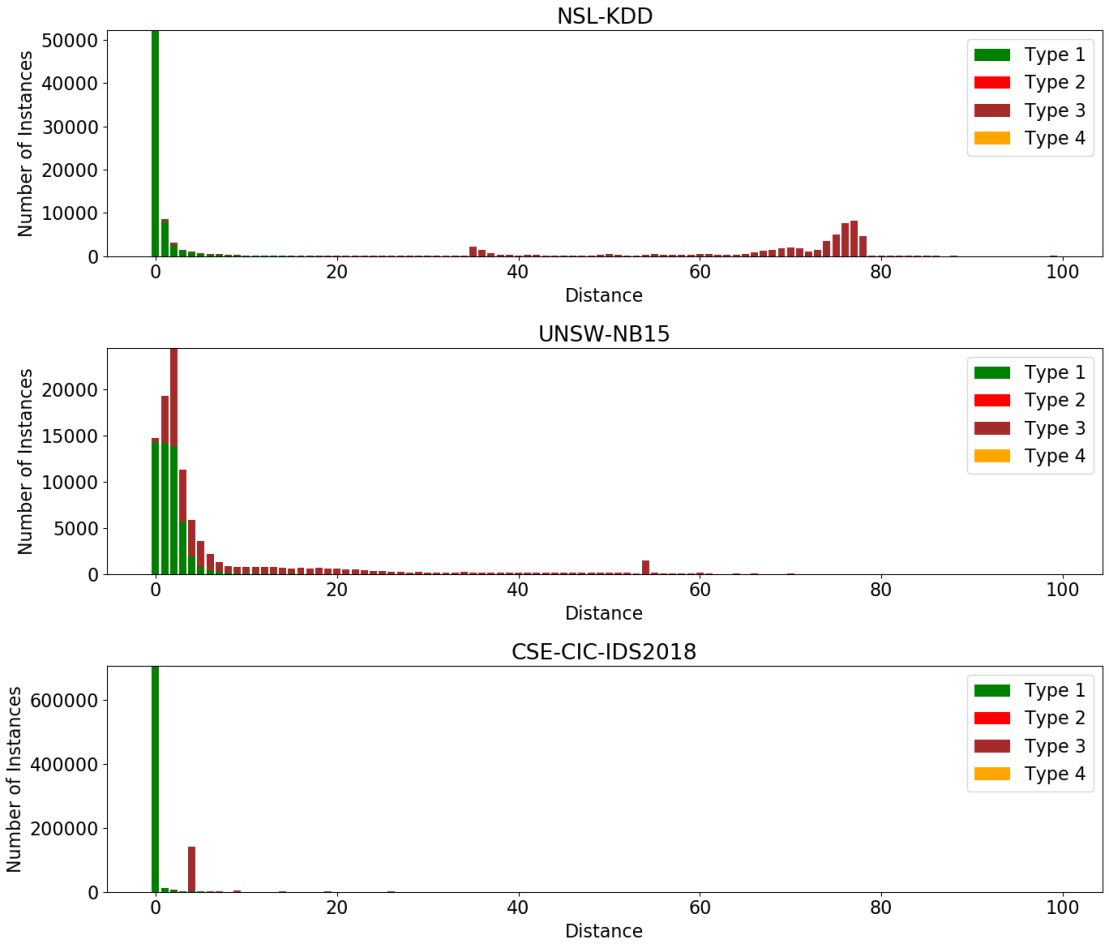

Figure 3.2: Distance of data instances to their nearest benign neighbor. On top the *NSL-KDD* dataset, in the middle the *UNSW-NB15* and on the bottom the *CSE-CIC-IDS2018* dataset is shown. Green represents type 1, red type 2, orange type 3 and brown type 4.

The next data basis is the distance of each instance to the next benign instance. The result is shown in Figure 3.2. Since only the nearest benign neighbor is considered, only instances of type 1 or type 4 are possible. Here, the distance is only divided into 100 quantiles and all quantiles and therefore all instances are displayed. With this data basis the assumptions made above can be mostly confirmed. In the *NSL-KDD* dataset benign and malicious clusters are mostly well separated. Only a few instances are near benign instances. The assumption that some benign and malicious clusters in the *UNSW-NB15* are really close to each other can be confirmed. Some of them are also well separated. In the *CSE-CIC-IDS2018* dataset the distance of different cluster types is not as large as expected. This means that they are distinct but the difference in the features for an instance

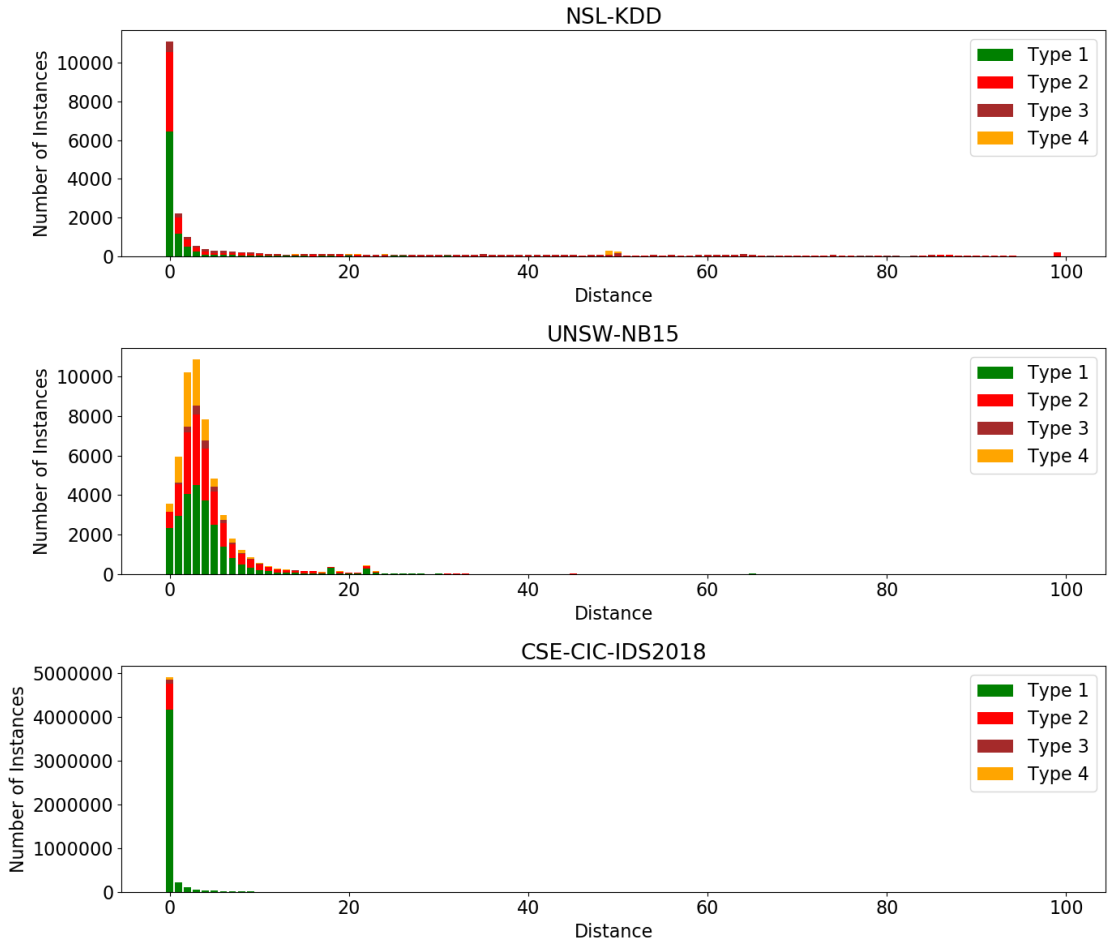

Figure 3.3: Distance of data instances for the test set to their nearest neighbor in the training set. On top the *NSL-KDD* dataset, in the middle the *UNSW-NB15* and on the bottom the *CSE-CIC-IDS2018* dataset is shown. Green represents type 1, red type 2, orange type 3 and brown type 4.

for being anomalous instead of being benign is small.

The next data basis is the most interesting one. Here, the distance of each instance included in the test set to the closest instance in the training set is calculated. The distances are divided into 100 quantiles and all quantiles are displayed. First of all, in every instance of each test data the distances are greater than zero, so no duplicates between training and test set exist. Then it can be seen in Figure 3.3 that each dataset shows different results. In the *NSL-KDD* dataset the difference to Figure 3.1 is not large, in both figures most instances are close to their neighbors. With 4310 instances of type 3 and 684 instances of type 4 the test set has overlapping clusters with clusters of another type in the training

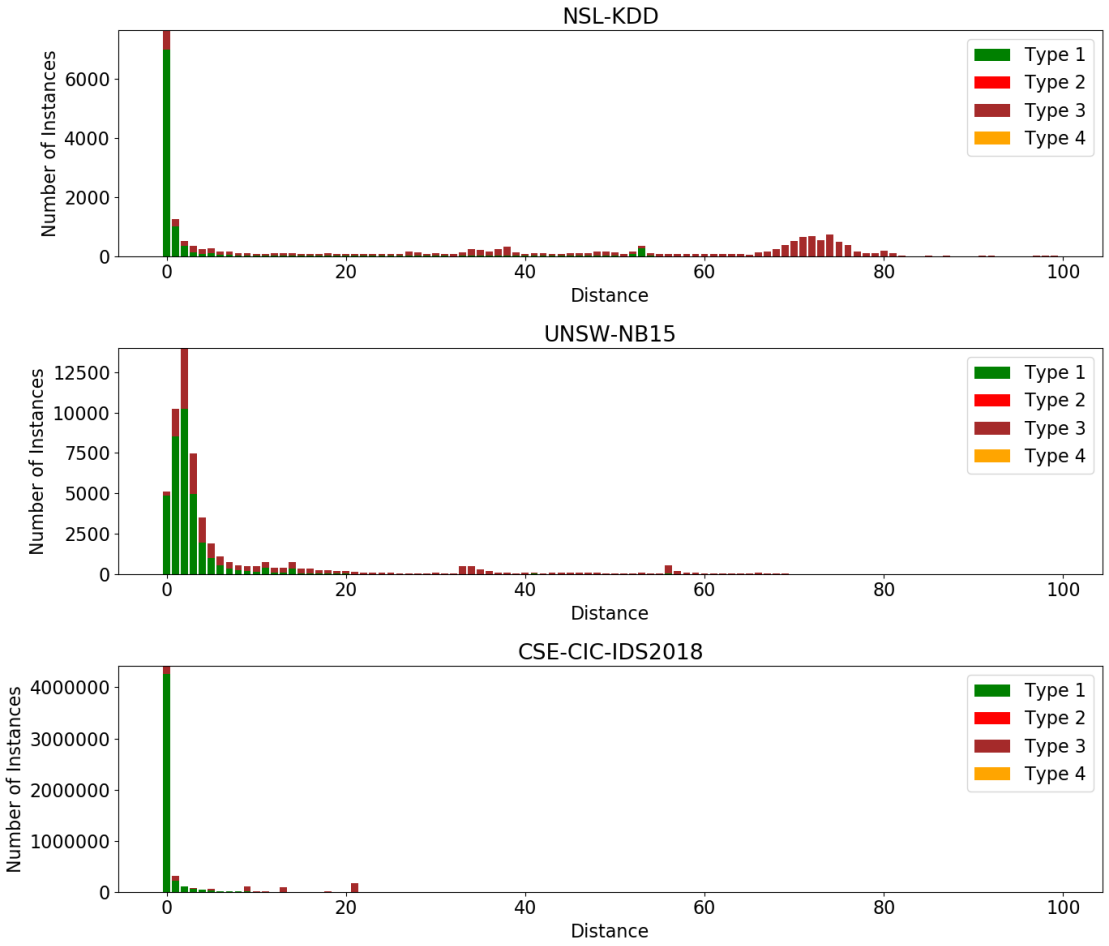

Figure 3.4: Distance of data instances to their nearest benign neighbor. On top the *NSL-KDD* dataset, in the middle the *UNSW-NB15* and on the bottom the *CSE-CIC-IDS2018* dataset is shown. Green represents type 1, red type 2, orange type 3 and brown type 4.

set. This indicates that there are fewer benign instances not being in benign clusters than malicious instances not being in malicious clusters. The *UNSW-NB15* behaves opposite to this. Here are 1814 instances of type 3 and 9523 of type 4. Many benign instances are near malicious ones.

In Figure 3.4 it can be seen that for the test set the distances of type 3 is not so large as it is in Figure 3.2. The biggest difference can be seen for the *CSE-CIC-IDS2018* dataset. In the training set an threshold exist that separates instances of type 1 with them of type 3. This threshold would separate the instances in the test set worse. In the *NSL-KDD* a cluster of benign data exists which is not represented in the training set.

The result of this analysis is that there are clusters in the datasets and that they can be separated. Only having the distance of an instance to it nearest neighbor as a metric limits the quality of classification. Also, the maximum distance varies between the datasets.

3.1.2 Removing Points

Data points of real world data have the characteristic to build clusters. When a cluster is really dense it can be represented with fewer points. The advantage is a faster training time and, depending on the used algorithm for the classifier, a faster classifying time. Besides, a bias towards this dense cluster can be removed. However, fewer points can also results in a worse classifying performance, because fewer training instances might not represent the full characteristic.

For the k-distance algorithm it is important that no gaps in a cluster are done. Gaps in clusters of a reduced data set can lead to the case that a benign instance is classified as an anomaly by the classifier. Also, a lot of instances in the data slows the classifier down. An ideal approach for solving these opposing problems has to be found. Because this reduction only needs to be done once, the performance of the reduction algorithm is not important.

First of all, all duplicated data instances that occurs in the datasets have to be removed. This can easily achieved by sorting all instances and then removing all instances, whose predecessor is equal. Removing only nearby points is more difficult. An algorithm for data reduction shall have a parameter that defines gaps of acceptable length between the points. With this parameter, every point builds a sphere with the radius equal to this parameter. Now, if every part of a sphere around point A is in another point's sphere, point A can be removed without tearing a gap. Therefore, when an instance shall be removed, it needs to be checked if its sphere is fully contained by other spheres. Figure 3.5 shows why a simple distance check with the help of a threshold is not sufficient. Not the distance to a point is relevant, but the combination of the closest neighbors. This problem needs to be solved for spheres in any number of dimension. For this a Voronoi diagram can be used. A Voronoi diagram divides a space into regions with a given set of seed points. The seed point of a region is the closest seed point to all points in its corresponding region. All points on the edge of a region have at least two closest seed points. Now, with a Voronoi diagram, it is simple to check if a point is necessary and whether it can be removed without tearing a gap. Therefore, the distance to all vertices of a region to its

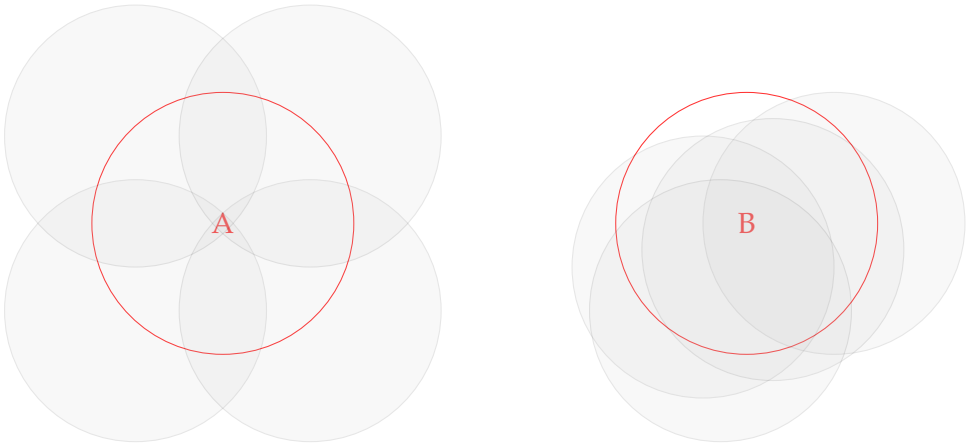

Figure 3.5: Sphere A is completely lying in the spheres of its neighbors. Even if the neighbors sphere around B were closer to B, it would not be fully covered by these spheres. Removing B would lead to a worse cluster boundary.

corresponding seed points is measured. If the maximum of these distances is larger than half of the parameter's value (t), removing this point could tear a gap, otherwise it does not.

These vertices are the points farthest away from the seed point. If a vertice's distance from a seed point exactly equals half of t, its distance from the neighbor's region seed point does also equal half of t. This means that the maximal distance between these two seed points corresponds to t. This is also true for each point of the edge of the region. Thus, the distance between each point that is lying on the direct line from a point on the edge to the seed point, is smaller than t. Since all points in an area are lying on a direct line from a point on an edge to the seed point, every point is closer to an adjacent seed point than t.

Algorithm 1: Pseudocode of the reduction algorithm

```
Data: lp list of points, radius r
Result: reduced list of points
sort Points;
for i ← 1 to length of lp do
    if lp[i] = lp[−1] then
        remove lp[i];
while a point was removed last loop do
    create Voronoi diagram vd out of lp;
    foreach point p in lp do
        dist ← max(vd junction points distance of p);
        if dist < r/2 then
            remove p from vd and lp
```

It might happen that this algorithm does not detect instances which can be removed without tearing a gap. However, each instance that is removed according to the algorithm shown above, is, without any doubt, removed without tearing a gab.

It is possible to extend the method. If a dataset shall be reduced to a fixed size, the radius can be increased after each iteration until the number of remaining points falls below the maximal boundary.

3.1.3 Incremental point insertion

Another method of reducing the number of data points is to start with an empty set of points and to add them step by step. A threshold is selected in order to determine how many points shall be included in the new set and thus when the algorithm is supposed to stop.

The first point f is the strongest outlier in the training set T, which is the point with the farthest nearest neighbor. This point is added to the set S. Then the point in T which is farthest away from f is added to set S. Until the size of set S hits the threshold, the furthest points in T to all point in S is added to S iteratively.

This approach reduces the bias to a dense cluster and keeps outliers.

3.1.4 Data augmentation

A different approach to 2.3.2 is to look at each instance and its neighbors. If there are too many data points in a specific neighborhood radius, no point is added. But if there are only a few data points, an additional data point will be added. The new data point position is randomly selected somewhere in the convex hull of data points in the specific neighborhood radius. The stochastic distribution in this convex hull can be selected arbitrarily.

This method adds a bias to clusters, but reduces the bias to denser clusters. Also, with the convex hull the chance that this type of augmentation will blur the transition between a benign cluster and a neighboring anomalous cluster is lower.

3.1.5 Anomaly Injection

In a dataset it might happen that huge areas are not filled with instances. Also, not known anomalies can occur in this areas. Depending on the classification algorithm, these areas can be identified as non anomalies because no point in the dataset in this area has been classified as anomalous. Also, if the dataset only contains non anomalous data points, anomalous points have to be added for some methods. It is desirable that the number of normal data points and the number of anomalous data points are almost equal.

A simple method is to fill empty regions with data points that are labeled anomalous. For each new point a random location is chosen. Then the distance to the closest neighbor is computed. If this distance is too small, a new random location is chosen. This is repeated until a random location is found where the distance to the nearest neighbor is sufficient. But if after a specified number of unsuccessful iterations for one point occurred, this point can be added if only the distance to the nearest non anomalous neighbor is sufficient. This ensures that this algorithm will terminate.

3.1.6 Removing Features

It is not necessary that all features of the data have a correlation with the related class. Also it is possible that this feature confuses a classifier. So removing features from data can result to a higher accuracy. Furthermore, a lower count of features results in a faster training and predicting computation time. This method requires that the whole dataset is labeled.

Selecting these removable features is not that easy. As shown in Figure 3.6, one fea-

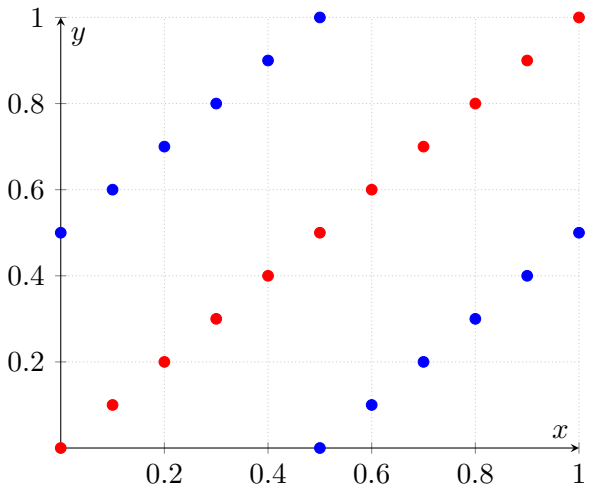

Figure 3.6: Data points in the two dimensional space. The two classes (red and blue) are easily distinguishable.

ture that seems to be useless for classification can be really useful in conjunction with another feature. For feature x and feature y it is not determinable to which class an instance belongs. Both classes are normally distributed over the complete feature space. But with both features together each instance can easily be assigned to the right class.

It is not practical to test whether all possible combinations result into a distinguishable space. This two dimensional example can be converted into a higher dimensional case. Thus, all combinations with all numbers of dimensions have to be tested. With n features it will yield to $(n-1)!$ possible combinations.

Therefore, another method for feature reduction needs to be used. The approach is to train different simple neural networks with all features of the train set. For each feature one run per neural network is made. Each run consists out of multiple iterations to reduce the influence of the variance in evaluating neural networks. The test set is the training set, but the feature under investigation is replaced by a random noise. The resulting accuracy for each run and iteration is compared to them with other features and also to a run without any manipulation of the test set. The features which do not causes any significant increase of the loss during evaluation can be removed.

Impact of Features

Different features have a different impact on the result. Some may include a strong random noise and will corrupt the classification quality. These features have to be detected and their weight in the classification must be reduced. Every approach is repeated ten times and each run uses six different neural networks, each with a different structure. Categorical cross entropy is used as loss function in order to compare the results. It is used because it is a non linear function and punishes a lower error with a logarithmic falling loss. The mean of all six neural networks and the ten iterations is used as final result. The full result can be seen in Table 2, the top ten features for each dataset in Table 3.1.

KDD Feature	Loss	UNSW Feature	Loss	IDS2018 Feature	Loss
hot	0.2422	dload	0.7346	Fwd Header Len	3.1721
wrong_fragment	0.1879	dttl	0.6642	Tot Fwd Pkts	1.6055
IRC	0.1347	ct_srv_dst	0.4874	Down/Up Ratio	1.6036
dst_host_rerror_rate	0.1190	synack	0.4815	Subflow Fwd Pkts	1.5321
private	0.1120	sloss	0.4074	Fwd Act Data Pkts	1.4362
count	0.1093	sctp	0.4003	Flow IAT Max	0.4814
urp_i	0.1089	arp	0.3708	Init Bwd Win Byts	0.3737
S1	0.1072	is_sm_ips_ports	0.3609	Active Max	0.3618
dst_host_srv_count	0.1050	igmp	0.3338	Active Mean	0.3345
RSTR	0.1028	sbytes	0.3300	Bwd IAT Tot	0.3267
default	0.0725	default	0.1479	default	0.0214

Table 3.1: Top ten features with the highest losses in the datasets. The features' meaning are explainded here [Feaa][Feab] [Feac]

This approach shows that some features have a very high impact on the neural network, whereas some have nearly none. The features and losses between the datasets cannot be well compared, because each dataset has different kind of features. Also, a high loss for the feature *IRC* in the *NSL-KDD* dataset indicates that this is an old dataset. It has to be determined whether reducing the input dimension by only using this features does have a positive impact on the classification results.

3.1.7 Dependent Transformer

The depended transformer is a transformer that checks the characteristic for each feature and does a transformation dependent on it.

Distribution of Features

For selecting the right transformer for each feature it is necessary to know how the values of each feature are distributed. This cannot be measured with a mean and a standard deviation because it might happen that a feature consists of more than one Gaussian distribution or even no Gaussian distribution at all. An example for this is when lots of values of a feature are close to zero or are close to one but only few between them.

So an algorithm that determines the type of distribution has to be found. In this thesis the types of distribution are defined as follows:

Gaussian Distribution A normal Gaussian distribution, or a distribution that can be approximated with a Gaussian distribution.

Binary Distribution In this distribution, only the values zero or one occurs. It is irrelevant how these two values are distributed, no method for a sensible transformation exits.

Single Edge Distribution Most values occur near one of the edges of the feature space. This is Very similar to a logarithmic or exponential distribution.

Double Edge Distribution Most values occur close to the edges of the feature space. This is the opposite of a Gaussian distribution.

Uniform Distribution The values are distributed uniformly over the feature space. A huge deviation of a standard uniform distribution is allowed. In fact, all distributions that do not fit one of the above definitions is treated as a uniform distribution. For example, multiple Gaussian distributions in one feature will be classified as an Uniform distribution, because a transformer would only have little impact on changing the distribution.

To determine the distribution, all features are tested separately in order to check whether they fit the respective distribution. The order of testing the distributions is essential, because a Binary distribution is also fitting the requirements of an double edge distribution. Therefore, it is first tested if the feature is binary. If it is not, it is tested for a Gaussian distribution with the Lilliefors test [Lil67]. This test checks whether a sample of data results from a Gaussian distribution even if the mean and the variance is unknown. Afterwards it is tested for a Single Edge distribution. This is done by checking if 90% of all samples have a distance smaller than 0.1 to one of edges. The Double Edge distribution is tested next. Here, 90% of all samples must have a smaller distance than 0.1 to one of the edges.

All features that are not classified by one of these methods are classified as a uniform

distribution.

The result of this evaluation is summarized in Table 3.2. It is noticeable that no Gaussian distributions occurs in all three dataset. With a visualization of each feature's distribution, this result can be proofed. there are also some apparent differences between the three datasets. In the *CSE-CIC-IDS2018* dataset less binary features and more Single Edge distributed features occur, and in the *NSL-KDD* dataset less uniform distributions are found.

	Gaussian	Binary	Single Edge	Double Edge	Uniform
KDD	0	90	16	9	7
UNSW	0	156	21	2	15
ids2018	0	18	40	3	17

Table 3.2: Counts of distribution types per dataset after normalization.

Selecting the Right Transformer for each Feature

All values of a feature X are in the range $[0, 1]$ In all non uniform distributions some areas have a high density, whereas others have a very low one. If in the density cluster a point exists, at which the label of the instances change, a classifier has to find this point with a high accuracy. If the high density cluster is stretched, the accuracy for finding the accurate position of this special point is less important, because the same distance error in a lower density cluster means that less instances are wrongly classified. This is illustrated in Figure 3.7 for a two dimensional context. In the left unscaled image both classes are very close to each other, in the scaled one on the right hand side the distances between both classes are greater.

Thus, a transformer has to transform each distribution in such a way that it will use the whole feature space. Even if in all three datasets a Gaussian distribution has not occur, it can occur in other datasets. Consequently, one also needs to find a transformer for this.

A transformer for a Gaussian distribution changes the mean to 0.5 and the standard deviation is set to $1/6$. With this standard deviation statistically 99.73% of all values lie within the range $[0, 1]$; all other values are set to zero or one by the algorithm.

$$gaussianScaler(x, X) = \begin{cases} 0 & \text{if } \frac{x-mean(X)}{deviation(X)} < -3 \\ 1 & \text{if } \frac{x-mean(X)}{deviation(X)} > 3 \\ 0.5 + \frac{1}{6} \cdot \frac{x-mean(X)}{deviation(X)} & \text{else} \end{cases}$$

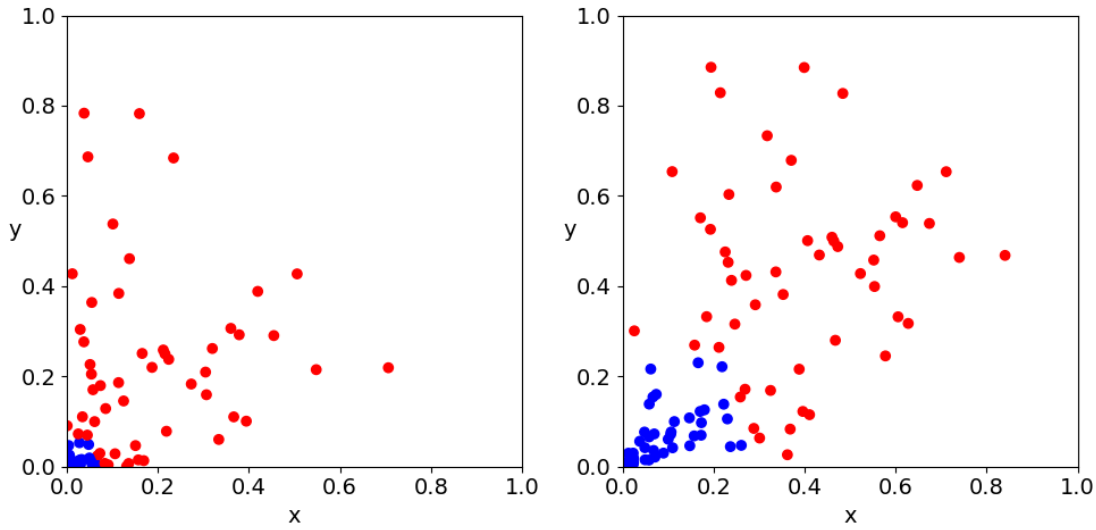

Figure 3.7: Left are random created unscaled instances, on the right they where scaled with $f(x) = \sqrt{x}$

A transformer for the Single Edge distribution stretches all points of an edge to a wider area:

$$LeftSingleEdge(X) = \sqrt{X}$$
$$RightSingleEdge(X) = 1 - \sqrt{(1 - X)}$$

For a Double Edge distribution this is done analogously:

$$DoubleEdge(x) = \begin{cases} \frac{\sqrt{X}}{2} & \text{if } x \leq 0.5 \\ 1 - \frac{\sqrt{(1-X)}}{2} & \text{if } x > 0.5 \end{cases}$$

Uniform distributions do not need any scaling, because they are already using the feature space efficiently. Binary distributions also do not need a transformer, because only two different values cannot be scaled usefully.

3.2 Combination of Approaches to One Classifier

Each method of preprocessing, dimension reduction and classification can be encapsulated in a module. Each module has a specific number of inputs, parameters and outputs. Some modules have an internal state which they have to train initially. The output of

one module can be the input of another module. So many modules can be chained in a pipeline to one classifier, which gets the raw input data and returns a score of classification after some computation. The modules and the order of modules can almost freely be chosen. The only condition is that the output dimension of one module matches the input dimension of the following module.

Some modules need a training phase to train their internal states. Therefore, each classifier needs two pipelines, one for training and one for classifying. After training all modules with an internal state give their state to the corresponding module in the classifying pipeline. The pipeline for training gets the whole dataset, whereas the classifying pipeline only gets one to n samples. A definition of a training pipeline also defines its corresponding classifying pipeline. Both are almost equal, except for modules that effect the size of a dataset, can only occur in the training pipeline.

3.2.1 Name Schema

In order to better understand the evaluation of which pipeline is used and which modules it contains, a precise name schema is used. Each module has its own short word and after the short word the used parameter for this module is listed. Modules are separated with an underscore. For example in a $NORM\_YJT\_PCA5\_KD$-pipeline the first module is a normalization module followed by a Yeo-Johnson transformer, a dimension reduction with PCA to five dimensions and uses the k-distance module as a final classifier.

One part of the evaluation is how well a pipeline classifies the data when only benign data is included in the training set. For this, FP at the first place in the pipeline name indicates that the training set only contains benign labeled data. Some of the modules such as PCA do not have an output in the range of $[0, 1]$. Thus, after these methods a normalization is made within the respective module, which is not mentioned in the name of the pipeline.

The modules and their short names are defined as follows:

Module Name	Point Remover
Module Class	Sample Reduction
Short Name	PR
Input Format	$N \times M$ where N is the number of samples and M is the dimension of a sample
Output Format	$N_r \times M$ where N_r is the new number of samples
Parameter	N_r, new number of samples
Needs Training	No
Description	Removes points from a dataset as described in 3.1.2. This module is only usable for the training pipeline. It reduces the bias to dense clusters.

Module Name	Incremental Point Insertion
Module Class	Sample Reduction
Short Name	IPI
Input Format	$N \times M$ where N is the number of samples and M is the dimension of a sample
Output Format	$N_r \times M$ where N_r is the new number of samples
Parameter	N_r, new number of samples
Needs Training	No
Description	Adds points from a dataset to a new dataset as described in 3.1.3. This module is only usable for the training pipeline. It reduces the bias to dense clusters.

Module Name	Random Noise Augmentation
Module Class	Data Augmentation
Short Name	RNA
Input Format	$N \times M$ where N is the number of samples and M is the dimension of a sample
Output Format	$N_a \times M$ where N_r is the new number of samples with $N_a \cdot a = N$
Parameter	a, augmentation rate
Needs Training	No
Description	Adds points a dataset as described in 3.1.3. This module is only usable for the training pipeline. It does not change the bias in the data.

Module Name	Convex Data Augmentation
Module Class	Data Augmentation
Short Name	CDA
Input Format	$N \times M$ where N is the number of samples and M is the dimension of a sample
Output Format	$N_a \times M$ where N_a is the new number of samples with $N_a = N \cdot a$
Parameter	a, augmentation rate
Needs Training	No
Description	Adds points a dataset as described in 3.1.3 within a convex hull. This module is only usable for the training pipeline. It changes the bias significantly to low dense clusters.

Module Name	Anomaly Injection
Module Class	Data Augmentation
Short Name	AI
Input Format	$N \times M$ where N is the number of samples and M is the dimension of a sample
Output Format	$N_a \times M$ where N_a is the new number of samples with $N_a = N + a$
Parameter	a, added number of anomalies i, maximum number of unsuccessful iterations
Needs Training	No
Description	Adds points a dataset as described in 3.1.5. This module is only usable for the training pipeline. It changes the bias significant to low dense areas.

Module Name	Normalization
Module Class	Normalization
Short Name	Norm
Input Format	$N \times M$ where N is the number of samples and M is the dimension of a sample
Output Format	$N \times M_a$ each value is in range $[0, 1]$ and $M_a \geq M$
Parameter	None
Needs Training	Yes
Description	Normalizes each value in a dataset as described in 2.3.3.

Module Name	Principal Component Analysis (PCA)
Module Class	Dimension Reduction
Short Name	PCA
Input Format	$N \times M$ where N is the number of samples and M is the dimension of a sample
Output Format	$N \times M_r$ where M_r is the new dimension with $M_r < M$
Parameter	r, the target dimension
Needs Training	Yes
Description	Applies the Principal Component Analysis to the data.

Module Name	Isomap
Module Class	Dimension Reduction
Short Name	ISO
Input Format	$N \times M$ where N is the number of samples and M is the dimension of a sample
Output Format	$N \times M_r$ where M_r is the new dimension with $M_r < M$
Parameter	r, the target dimension
Needs Training	Yes
Description	Applies the Principal Component Analysis to the data. Memory consumption is with $\Theta(N^2)$ very high.

Module Name	Autoencoder
Module Class	Dimension Reduction
Short Name	AE
Input Format	$N \times M$ where N is the number of samples and M is the dimension of a sample
Output Format	$N \times M_r$ where M_r is the new dimension with $M_r < M$
Parameter	r, the target dimension
Needs Training	Yes
Description	Trains an autoencoder and transforms all input data with the trained encoder.

Module Name	Feature Removing
Module Class	Dimension Reduction
Short Name	FR
Input Format	$N \times M$ where N is the number of samples and M is the dimension of a sample
Output Format	$N \times M_r$ where M_r is the new dimension with $M_r < M$
Parameter	r, the target dimension
Needs Training	Yes
Description	Checks the influence of each feature as described in section 3.1.6. This module only keeps the top r features with the highest influence.

Module Name	Quantile Transformer
Module Class	Transformer
Short Name	QT
Input Format	$N \times M$ where N is the number of samples and M is the dimension of a sample
Output Format	$N \times M$
Parameter	None
Needs Training	Yes
Description	Transform each feature to a uniform distribution.

Module Name	Yeo-Johnson Transformer
Module Class	Transformer
Short Name	QT
Input Format	$N \times M$ where N is the number of samples and M is the dimension of a sample
Output Format	$N \times M$
Parameter	None
Needs Training	Yes
Description	Transform each feature with the method of Yeo-Johnson.

Module Name	Dependent Transformer
Module Class	Transformer
Short Name	DT
Input Format	$N \times M$ where N is the number of samples and M is the dimension of a sample
Output Format	$N \times M$
Parameter	None
Needs Training	Yes
Description	Transform each feature depending on the distribution type of the feature.

Module Name	k-distance
Module Class	Classifier
Short Name	KD
Input Format	$N \times M$ where N is the number of samples and M is the dimension of a sample
Output Format	$score_{benign}, score_{anomaly}$, the score of the classifier with $score_{benign} + score_{anomaly} = 1$ and $score_{benign}, score_{anomaly} > 0$
Parameter	k, the number of considered nearest neighbors for this score d, the distance when an instance is classified as anomaly
Needs Training	No
Description	Considers the k nearest neighbors for each instance and their distance to this instance. Details are described in 3.1.1.

Module Name	Neural Network
Module Class	Classifier
Short Name	NN
Input Format	$N \times M$ where N is the number of samples and M is the dimension of a sample
Output Format	$score_{benign}, score_{anomaly}$, the score of the classifier with $score_{benign} + score_{anomaly} = 1$ and $score_{benign}, score_{anomaly} > 0$
Parameter	s, the structure of the neural network
Needs Training	Yes
Description	Trains a neural network.

Because a structure of a neural network is not a short parameter, five different struc-

tures are defined and are referenced by a number. For example, $NN2$ uses the structure defined by 2 in the Appendix. Two neural networks with the same defined structure do not need to have an equal number of trainable parameters. Besides the structure, the number of features is also important for this and this vary by the output dimension of the module before the neural network module.

4 Implementation

4.1 Choosing the Programming Language

There are a lot of programming languages that can be used for this thesis. However, the most suitable programming language for this thesis is Python. This has different reasons. First of all, Python has a large community so there are a lot of useful open source libraries, a good documentation and also lots of code examples. It has a good performance on numeric and matrix operations. Additionally, it runs on many operating systems and on different hardware architectures. With reasonable time investment Python's main disadvantage can be eliminated. These are its bad performance on (conditional) jumps and problems related to dynamic typing, both disadvantages of script languages in general. They can be eliminated with Cython, which extends Python but strictly represents a separate programming language. In Cython variables can have C types besides Python types. They are used to get a Cython code which can be compiled to C to get a speed up.

Another approach is to use the Julia language. It has a syntax that is very similar to Python, but allows static typing. During runtime it also compiles to C code and it allows to run Python code as sub modules. Basic Python types can be handled and manipulated by Julia. Julia can be more secure than Python if static typing is strictly used. However, converting Python code to Julia is more time consuming than converting from Python to Cython.

Thus, Python is the best programming language for this thesis since it represents a good basis. In case it does not meet the performance requirements, the relevant parts can be sourced out to Cython or Julia.

4.2 Additional Libraries

In this thesis different external libraries are used for the implementation.

4.2.1 Scikit-learn

Scikit-learn is a free open source library for Python focusing on machine learning algorithms [PVG+11]. It has a good documentation and it is well maintained. It is under

BSD-License.

4.2.2 TensorFlow

TensorFlow is a free open source library for neural networks developed by Google [AAB+15]. It is licensed under Apache License 2.0. Its main API is for Python but APIs for other language do also exist.

Keras

Keras is a free open source high level API [C+15]. Its goal is to deliver a high level API for different neural network APIs such as TensorFlow, CNTK, or Theano. Keras is official part of TensorFlow 2.0. In this thesis Keras is used as high level API for TensorFlow.

4.3 Adjustment of Existing Implementations of Used Algorithms

Some existing algorithms can have problems with the characteristic of network data. However, this does not mean that they are not suitable for this area of application because small changes in the implementation can eliminate these flaws.

Isomap

In this thesis the Isomap implementation of scikit-learn[4] is used. The problem with the Isomap algorithm is that multiple clusters result in a bad low dimensional embedding because it uses geodesic distance instead of euclidean distance. Geodesic distance is computed by generating a k-neighbors graph with euclidean distance between neighboring points. The geodesic distance is the shortest path between two points in the k-neighbors graph. However, it is likely that two clusters are not connected in the graph. If this is the case, problems arise because the distance between the points in one cluster do not have a defined distance to the points in another cluster. The implementation of the scikit-learn package sets this distance to zero. This way, each point in a cluster have a zero distance to each points in other clusters. This result into an embedding that comes close to an infinity embedding error. A solution for this problem is to connect all clusters in the graph. Here, the difficulty is which clusters shall be connected to each other. In order to determine which specific point of a cluster shall be connected to which specific point in another cluster needs seven steps. In the following X is the set of all points, C is the set of all clusters and $distance$ is the euclidean distance between two points.

[4] https://scikit-learn.org/stable/modules/generated/sklearn.manifold.Isomap.html

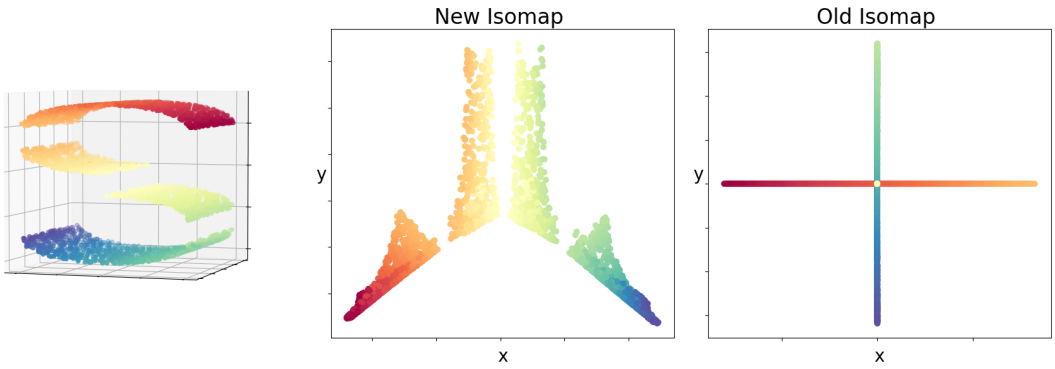

Figure 4.1: A manifold with four clusters cannot be embedded well by the Isomap algorithm. With a small adjustment of the algorithm it can reach a suitable embedding error.

1. Relate each point $x \in X$ to a cluster c
2. Find $p_{i,j}$ for $i, j \in \{1, ..., |C|\}$ where $i \neq j$ and $p_{i,j} = X[argmin(distance(\forall p \in c_i, \forall p \in c_j))]$
3. Create a new graph g out of all points found in step two. All points are connected to each other.
4. Update the weight matrix in graph g with the following rule:

$$w_{m,n} = \begin{cases} 0 & \text{if cluster of } p_m = \text{ cluster of } p_n \\ distance(p_m, p_n) & \text{if cluster of } p_m \neq \text{ cluster of } p_n \end{cases}$$

5. Find the path in g with the lowest weight from c_i to c_j for $i, j \in \{1, ..., |C|\}$ and $i \neq j$
6. For each step in the path add the two points as a pair to a list l if they are in different clusters and if this pair is not included in l yet
7. Add each pair in l as a new edge with the euclidean distance as weight to the original k-neighbors graph

Afterwards, the distance matrix is calculated from the k-neighbors graph as it is normally done.

With this adjustment it is possible that two clusters are directly connected even if a newly created connection has connected them indirectly. This has two advantages: First, the clusters are not connected in a chain and second manifolds are not broken up. In

the following a ring manifold consisting of several clusters is considered. If the rule of connecting clusters is that only two clusters are connected directly if there are no other direct or indirect connections this will lead that the ring manifold will not be connected to a ring, instead one gap will still exists. Thus, it is important to determine the length of the indirect connection and to decide whether a new direct connect between these two clusters should be made.

In Figure 4.1 the advantage of the adjustment is clearly visible. The original Isomap algorithm cannot detect the manifold in the data. The adjusted algorithm creates three new connections and therefore the geodesic distance between points of different clusters can better be approximated.

Quantile Transformer

The disadvantage of the scikit-learn[sklb] implementation of a quantile transformer is, that is does not eliminate unique values within one feature. If a value occurs more often than the size of one quantile allows, the next quantile is left empty. Here the adjustment is really simple. When the Quantile Transformer is fitted, values that occur more than once are reduced to one occurrence. In Figure 4.2 the original data is transformed to a uniform distribution with the adjusted transformer. The reason why the original transformer does not create a good uniform distribution is simple. The duration feature of the *UNSW-NB15* dataset has a lot of zeros and a lot of near zeros values which occur multiple times. The huge amount of zeros blocks the first quantiles and leaves them empty, the amount of equal near zero values leads to sparsely fitted quantiles.

4.4 Challenges in Implementing Modules

Not all of the previous defined modules can be programmed straightforwardly. Special cases as well as the performance and memory consumption of the implementation have to be considered.

4.4.1 Incremental Point Insertion

The Incremental Point Insertion algorithm has the problem of a bad performance scaling with a growing number of instances in the dataset. An naive implementation where every point is checked between all other points will have a runtime of $\mathcal{O}(t \cdot (t \cdot n \cdot k))$ where n is the number of instances, k the number of features and t the target size of instances. For each incremental insertion (step of iteration with t total steps) the distance from each

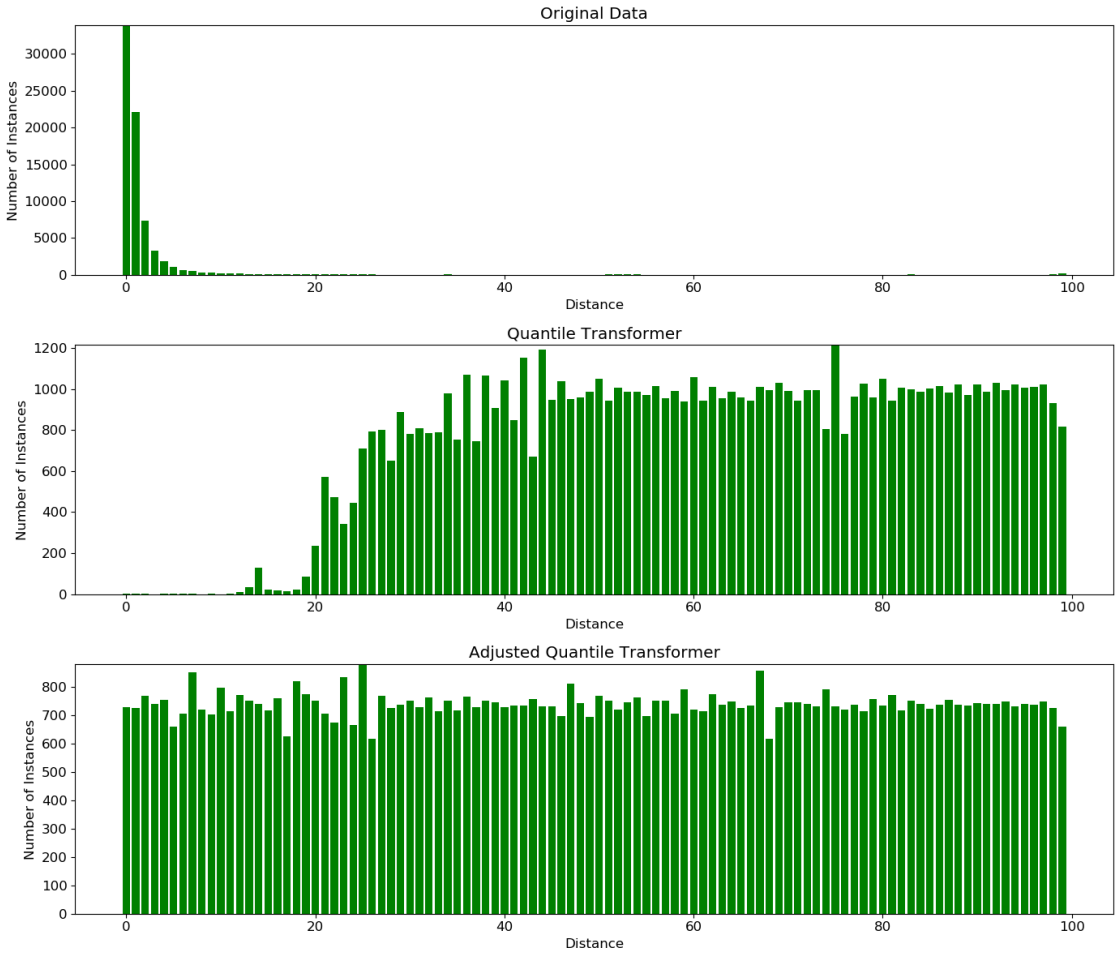

Figure 4.2: The duration feature of the *UNSW-NB15* dataset after eliminating multiple occurrences of one value. The adjusted transformer achieves a better uniform transformation than the original one.

point in the reduced set (zero to t points, grows with each step by one) to each point in the original set (n to $n - t$ points, decreases with each step by one) is computed. The computation of the distance is linear to the dimension. The needed point with furthest nearest neighbor can be detected on the fly during the distance calculation and is then inserted in the reduced set.

With a kd-tree the search for the nearest neighbors can be reduced to $\mathcal{O}(n \cdot log(t))$ [FBF77], but the used scikit-learn implementation for the kd-tree does not support inserting new points to the tree, thus the kd-tree must be rebuilt in each step completely. So in each step a construction time of $\mathcal{O}(t \cdot (k + \log t))$ must also be considered. All in all the complexity

is:

$$\mathcal{O}(t \cdot (n \cdot \log(t) + t \cdot (k + \log(t)))$$

In fact, the kd-tree can be built faster if the reduced set is sorted by the first feature every time a new point is added. Because the reduced set is almost presorted, an algorithm such as Timsort can sort it in $\mathcal{O}(t)$ time. So the kd-tree implementation, which uses Timsort, does not need much time for sorting the first feature. In fact, querying the tree takes much more time than building it because n is much larger than t.

An other optimization is to reduce the number of points which are querying the kd-tree. After finding the first point for the reduced set, the distance from all points in the original set to this point is computed[5]. Afterwards, a mask is used. All points that have a distance to their nearest neighbor in the reduced set which is larger than a threshold remain unmasked. The mean is used as a threshold because it can be computed fast. When at each step a point is added to the reduced set, all points that are masked cannot increase their distance to the nearest neighbor, they can only decrease it. After adding this one point the distances of all unmasked points are recomputed. If the maximum nearest neighbor distance is larger than the mean of the previous step, it is certain that no masked point has a nearest neighbor that is farther away. Then the mask and mean are stored and a new more restrictive mask with a new mean is used for the next step. The new threshold for masking points is the maximum out of the mean and the previous mask's threshold. This is done until a recomputed distance to a nearest neighbor is lower than the current threshold. Then the last mask with a smaller threshold than the current mean is used, all discarded masks and their corresponding thresholds are deleted. If no mask has a smaller threshold; the mask is reset to unmask all points and the threshold is set to zero. So basically, each step consist of making the mask more or less strict.

This masking technique has a huge effect on the performance. Reducing a set of $100,000$ 3-d points to $1,000$ points takes $122.29s$ without this masking technique and $20.31s$ with it. Reducing $100,000$ 3-d points to $5,000$ points takes without a mask $769.01s$ and $96.47s$ with a mask. The masking technique has a strong impact on the performance of this algorithm, because it reduces the points that are querying the kd-tree significant.

Figure 4.3 shows the time it takes each step to find the nearest neighbors. It can be

[5] It is not important to edit the original set after each step, not doing this will only lead to a non empty intersection between the reduced set and the original set. Therefore, all points in the intersection have a distance of zero to the nearest point in the reduced set and are never be considered to be added to the reduced set again.

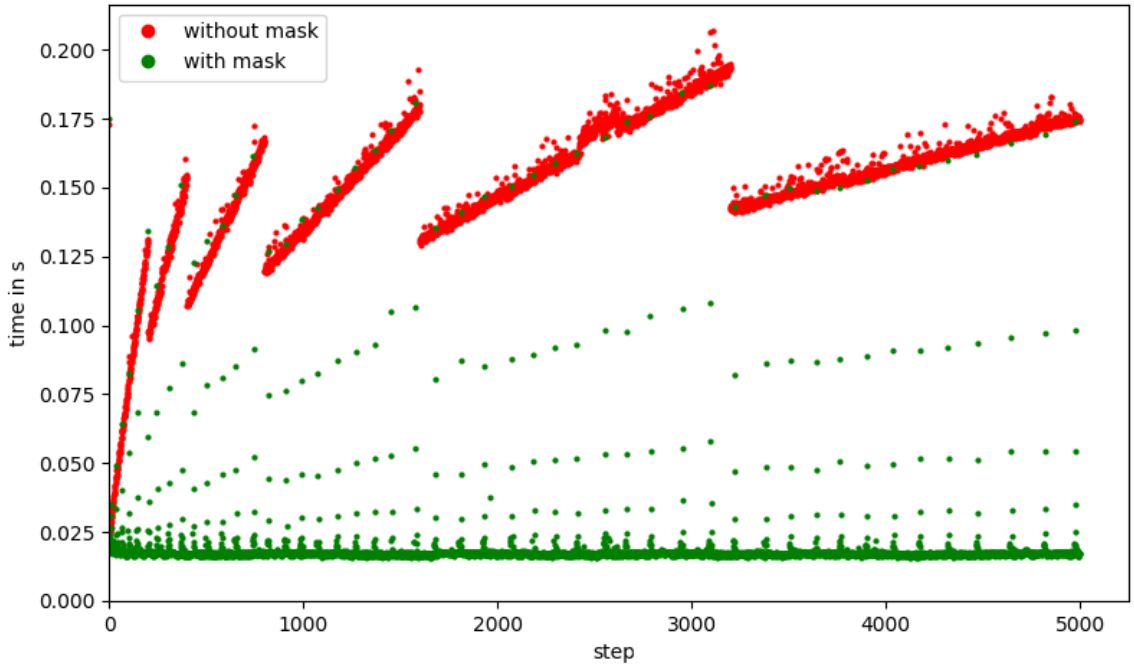

Figure 4.3: The time it takes each step to find the nearest neighbor with and without the masking technique.

seen that the mask is reset only a few times in comparison of $5,000$ total steps. Most steps with a mask have a computation time below $25ms$ whereas the steps without a mask have a time longer than $100ms$. Also, a large overhead for computing the mask cannot be seen in this figure. It is striking that the computation time makes jumps around certain points. The reason behind this might be that at a specific size the kd-tree adds new nodes or increases its depth, so that the query is more efficient. The figure only seems to show that the analytically determined $\mathcal{O}$ of the computation time is not valid. First of all, it is only an upper bound, secondly n is much larger than t, so the kd-tree building time is much faster than the query time. What can be seen is the $\mathcal{O}(n \cdot \log t)$ of the query with a constant n and a t that is growing on each step.

4.4.2 Anomaly Injection

The Anomaly Injection module deals with two problems. The first problem is to determine the best minimum distance of new anomalies to their nearest neighbor. It is appropriate to use a value that can be computed from the training dataset. This might be the median or mean distance to the nearest neighbor, or a distance based on them.

In order to find the best minimum distance, the distance from each point to its near-

est neighbor is computed and stored in a list. This list is sorted and then eleven values are extracted from it. These are the minimum, the maximum, and $(i/10 \cdot \#number\ of\ points)$th point with i from 1 to 9. Each of the three datasets is evaluated individually. In the training data all anomalies are removed and the training data is reduced to 10,000 instances to achieve a faster evaluation. Then for each iteration 50,000 anomalies are added by this module with the eleven different distance parameters. With this points, five equal, very simple neural networks are trained and is evaluated against the test data of a dataset. So all in all 165 runs are made. The mean for precision and recall out of this five runs is displayed in Figure 4.4. The evaluation does not show any significant influence of

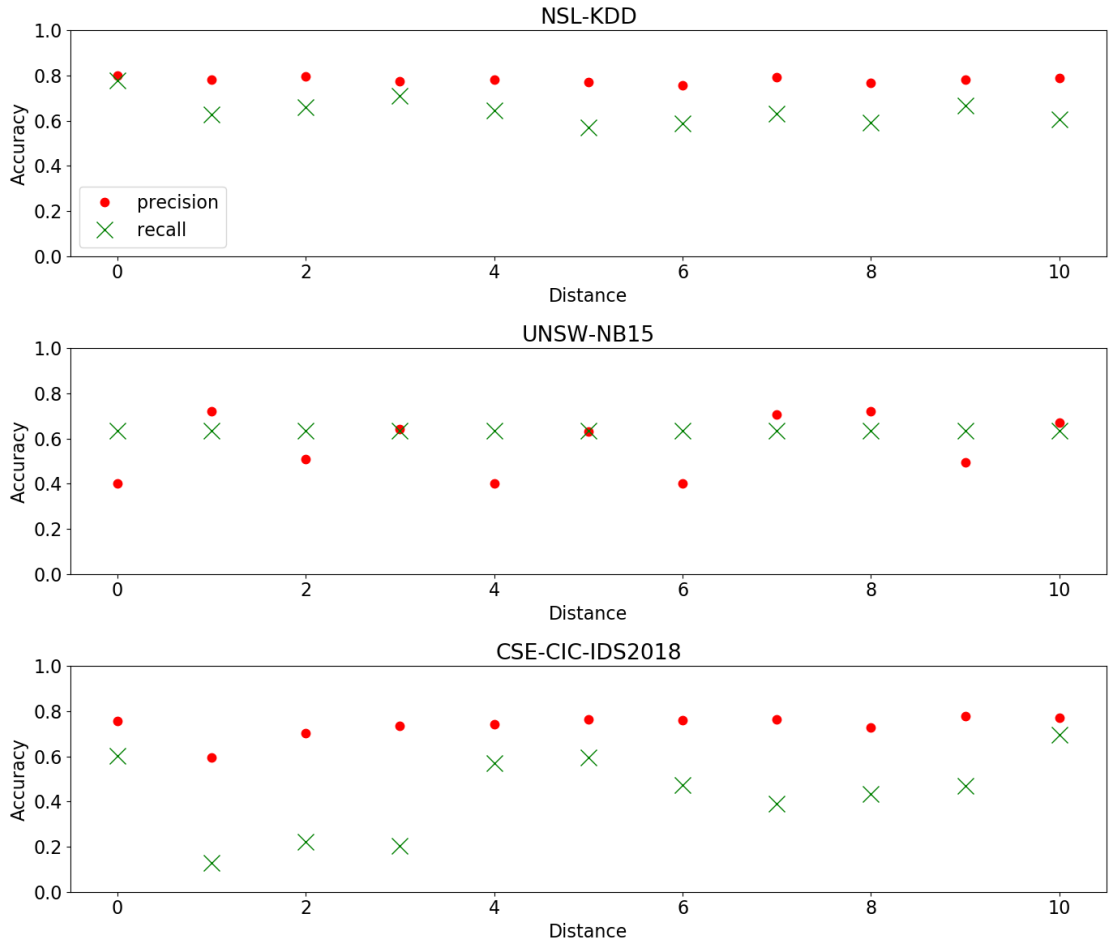

Figure 4.4: Precision and recall (y-axis) after Anomaly Injection with the *NSL-KDD* , *UNSW-NB15* and *CSE-CIC-IDS2018* dataset in dependence of the distance (x-axis). Precision and Recall are computed by the *evaluation_report* function of the Scikit-learn library.

the distance parameter. It seems that there is no sweet spot for a perfect distance, in the

CSE-CIC-IDS2018 dataset a larger distance seems to have a slight advantage. Besides, most results show a better predicting than chance, which means that the Anomaly Injection module is working. The augmentation factor of at least five is necessary because otherwise the neural networks were able to distinguish real data from augmented data. This results from impossible combinations of features due to the conversion of a categorical feature to multiple binary features. A random data point can have the *UDP* feature and the *TCP* feature both to one at the same time. Any real data does not have this combination and thus it is easy for a neural network to distinguish such values. However, since such data points are obviosly anomalies, it is necessary to generate such points.

The high dimensional space represents another challenge. There are so many possibilities for a random point, that most added points already have a large distance to their nearest neighbor. Thus, the distance parameter has no effect on the result. Therefore testing needs to take place in a smaller dimensional space. This reduced space is made with PCA and the new dimensionality is set to three. The result is shown in Figure 4.5. Here, the distance does not make a huge difference either, for each distance precision and recall are almost equal. So in future this distance is set to the median value and no other different parameters will be tested. However, an other observation can be made. The dimension reduction improves the classification result significantly for all three datasets. Thus, even if a lot of information is lost due to the dimension reduction, the advantage of filling a lower dimensional space with anomalies predominates.

The second problem of this module is its computation speed. This is solved by using multiple kd-trees for finding the nearest neighbor. One kd-tree has all benign training data, a second one, if present, all training anomalies. That the training data is contained in two trees results from the algorithm itself. If an anomaly injection was not successful multiple times because of an anomalous nearest neighbor, only benign nearest neighbor would be considered next time. A third kd-tree is for the generated anomalies.

4.4.3 Convex Data Augmentation

The main feature of the Convex Data Augmentation is the the convex hull. In order to build this hull the scipy Delaunay algorithm is used[6].

However, first of all it has to be determined which subset is used for the convex hull. For this, a start value d is chosen. Then for each point in the training set, all neighbor points with a distance smaller than d are determined. If a point p has more than u and less

[6]https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.Delaunay.html

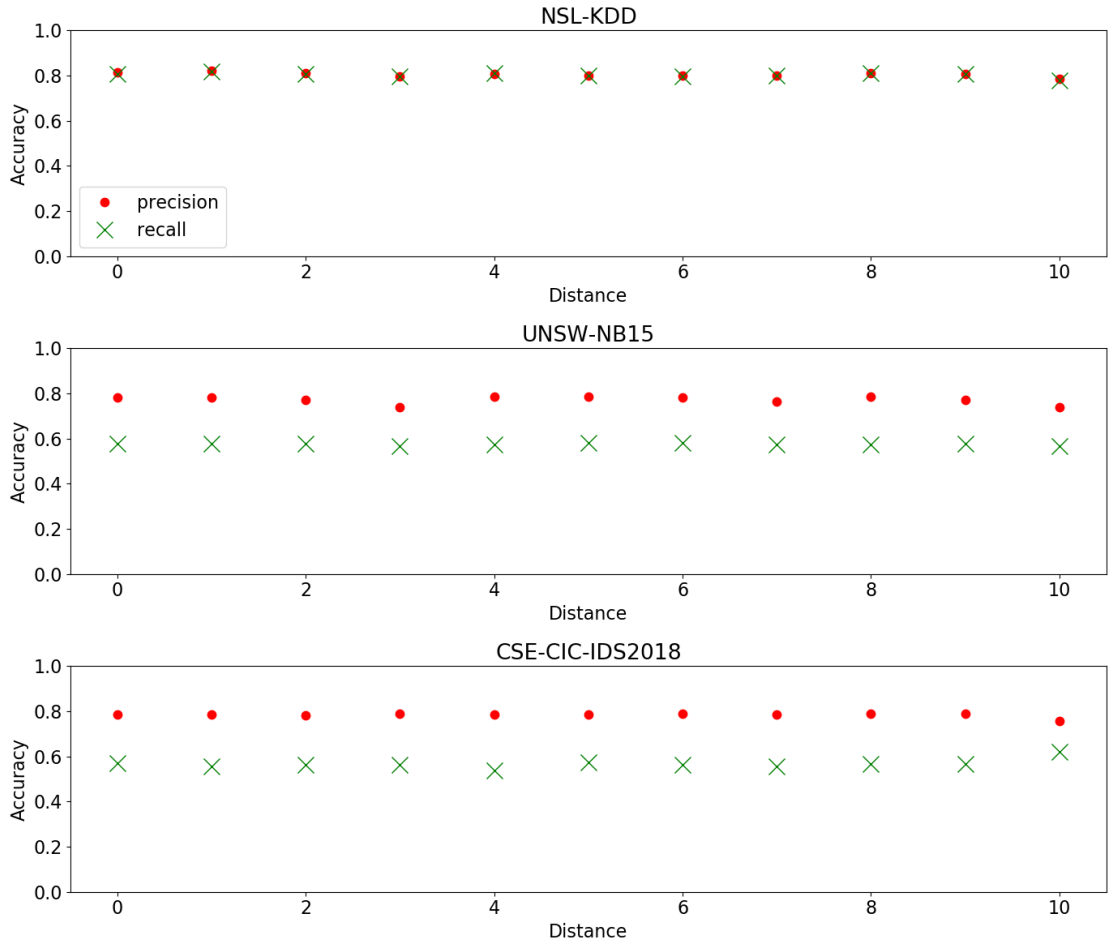

Figure 4.5: Precision and recall (y-axis) after Anomaly Injection of the *NSL-KDD* , *UNSW-NB15* and *CSE-CIC-IDS2018* dataset in dependence of the distance (x-axis) in a three dimensional space. Precision and recall are computed by the *evaluation_report* function of the Scikit-learn library.

then v of these nearest neighbors, p and all points closer than d are used for the convex hull. Then for each convex hull a point is added somewhere in the hull. If not enough convex hulls are found, the value d will be adjusted in the next step. If too many hulls are excluded by v, d is decreased. If too may hulls are excluded by u, d is increased. These steps are repeated until the required number of new points is added to the original set.

The problem of this module is to build the convex hull. In order to build a convex hull a Delaunay triangulation is made. For this, one needs at least as many points as dimensions are present. If all points have a feature in which all values are equal, this feature can temporally removed to reduce the dimension. But the problem that occurs

is that the scipy implementation cannot compute the convex hull with the given data. It raises following error:

> *scipy.spatial.qhull.QhullError: QH6154 Qhull precision error: Initial simplex is flat (facet 1 is coplanar with the interior point)*

In the description of the error message it says *The input to qhull appears to be less than 17 dimensional, or a computation has overflowed.* Although this error message does not appear every time, it occurs in most cases, even if u is set to a value larger than 100. After filtering all dimensions with zero difference in a feature, this error messages raises. So it seems that the characteristic of the data is not usable for a Delaunay triangulation. Therefore, another way to compute the convex hull or a different algorithm with a similar result has to be used.

This is done by using randomized weights for the normalized direction vector from p to each neighbor. After summing them all up, it is likely that the resulting vector r has a bias to a region with more points. For example, if the point p is on the edge of a cluster and most of the neighboring points are closer to the cluster center then itself, it is more likely that r will faces towards a direction closer to the cluster center. Next the maximum distance is computed. This is the maximum length of the direction vectors with an angle lower than $45°$ in respect to r. If there is no such direction vector the maximum length is set to half of the maximum length of all direction vectors. Afterwards, a random value between zero and this maximum length is chosen. So the new point's position is the position of p plus normalized r multiplied with the chosen random value.

4.4.4 Point Reduction

In this module the same problem as in 4.4.3 occurs. This module needs a Voronoi diagram which is the dual graph to the Delaunay triangulation. The standard procedure is to compute the Voronoi diagram out of the Delaunay triangulation. It seems that the algorithm does not cope well with features which have a low variance or which are binary features. Even if a full dataset with more than $100,000$ points is used, the algorithm complains that the input data does not appear to be fully dimensional. Because an algorithm for the reduction of points does already exits and since there is no obvious solution for this problem, this module will not be implemented in this thesis.

5 Evaluation

In this chapter different pipelines are evaluated. For evaluating a pipeline two aspects are important. First the validity of a pipeline: If the result cannot be trusted, this pipeline is more or less useless. The second aspect includes performance and hardware requirements. If the pipeline uses to much hardware resources it can happen that the incoming traffic cannot be processed. Other programs that are running on this machine can also be impaired.

In the following only remarkable results are addressed, a list including the results from other pipelines can be found in the appendix. Besides, the best out of the five neural network schema is shown.

5.1 Result of the most simple Pipelines

To measure the influence that a single module has on the classification report, the basic results have to be determined. This is done by using the simplest pipelines that only consist out of the normalization module and a classifier module. By comparing other modules to these simplest pipelines, a positive or negative effect can be shown. In Table 5.1 all basic results are listed. It can be seen that the neural network has a slight advantage over the k-distance algorithm in most of the metrics. The characteristic of the classification is also similar. This can be seen in the values of the True-Positive and True-Negative. In the *NSL-KDD* and *CSE-CIC-IDS2018* dataset both methods have a high True-Negative, in the *UNSW-NB15* both have a high True-Positive. The values also show why other prepro-

Pipeline	Dataset	Accuracy	TPR	TNR
Norm_NN3	KDD	81.66%	72.99%	93.1%
Norm_KD	KDD	79.5%	69.37%	92.9%
Norm_NN4	USNW	81.83%	96.75%	73.23%
Norm_KD	UNSW	78.84%	90.83%	71.92%
Norm_NN5	IDS	97.72%	83.06%	99.86%
Norm_KD	IDS	96.33%	87.23%	97.66%

Table 5.1: Evaluation results of the basic pipelines. All values are given in percentage. The difference between neural networks and K-distance is small.

Pipeline	Dataset	Accuracy	TPR	TNR
Norm_PCA5_NN1	KDD	77.85%	63.53%	96.78%
Norm_PCA5_KD	KDD	84.05%	78.14%	91.85%
Norm_PCA5_NN4	UNSW	74.05%	99.03%	59.64%
Norm_FR5_NN5	UNSW	69.39%	98.91%	52.36%
Norm_AE5_NN5	KDD	81.88%	74.67%	91.41%
norm_ipi10000_ISO_KD	IDS	93.57%	76.79%	96.02%

Table 5.2: Evaluation results of the four dimension reduction modules. All values are given in percentage.

cessing steps are necessary. A True-Negative of 73.23% means a False-Positive of 26.77% which is the rate of benign instances that is interpreted as malicious traffic. So at least every fourth instance would generate a false alarm. This effect might be reduced by using the frequency estimation method introduced in 2.4. However, it is unlikely that even with this method false alarms can be reduced significant without reducing the detection rate.

5.2 Dimension Reduction

Reducing the dimension of the input data involves an embedding error. However, this error can be small enough so that the classification result does not become significantly worse than the original. Also, the chance of overfitting is reduced and therefore the classification result might be better. To verify this, all four methods of dimension reduction are tested. The target dimension is set to five, because a lower dimension means a higher embedding error and with a high target dimension the impact of this embedding error will be lower and harder to measure. As the only module, the Isomap module gets a training set that is reduced to $10,000$ instances because otherwise the memory consumption will be too high to be handled probably.

The dimension reduction has different effects. First of all, reducing the data to five dimensions either leads to a low change in the classification result or a high one. In fact, PCA can improve the classification result for the *NSL-KDD* dataset by 4.55 percentage points, whereas for low embedding it reduces the classification error only slightly. The Autoencoder also shows some good results but sometimes it produces a very bad embedding so that in these cases the classifier is not better than chance. Removing a lot of features leads to the worst performance. Here the classification results are much lower than without the removal.

Pipeline	Dataset	Accuracy	TPR	TNR
norm_QT_KD	KDD	65.81%	71.07%	58.85%
norm_DS_KD	UNSW	80.09%	93.84%	72.16%
norm_YJ_KD	KDD	84.54%	78.90%	91.99%
norm_QTT_NN5	IDS	97.82%	83.73%	99.87%
norm_YJS_NN4	UNSW	81.68%	96.32%	73.23%

Table 5.3: Evaluation results of the three transformer modules. All values are given in percentage.

Pipeline	Dataset	Accuracy	TPR	TNR
norm_ipi10000_KD	KDD	78.31%	67.96%	91.99%
norm_ipi10000_NN5	KDD	79.28%	66.5%	96.18%
norm_ipi10000_AI3_KD	KDD	84.39%	75.25%	96.46%
norm_ipi10000_NN4	UNSW	74.92%	98.53%	61.29%
norm_ipi10000_KD	UNSW	77.47%	90.25%	70.09%
norm_ipi10000_NN5	IDS	94.59%	75.96%	97.31%

Table 5.4: Evaluation results of the Incremental Point Insertion module. The training datasets where reduced to 10,000 points. All values are given in percentage.

This shows that PCA represents the best method for dimension reduction and that Autoencoder as well as Feature Reduction are not working well.

5.3 Transformer

The Quantile Transformer does not have a positive effect on the classification result. With a reduction of the accuracy to 65.81% in the *NSL-KDD* dataset it causes a huge difference. Besides, the true negative is really bad, more than each third benign instance is classified the wrong way. In the other datasets this difference is not as strongly pronounced. The Yeo-Johnson transformer barely makes a difference when using neural networks as a classifier, but with the k-distance it can improve the classification result by up to four percentage points. The Dependent Transformer behaves similar to the Yeo-Johnson transformer but its improvement when using k-distance is only up to two percentage points.

5.4 Incremental Point Insertion

The Incremental Point Insertion module can achieve good results. When only $10,000$ instances of the original dataset are used, the drop of accuracy is negligible. With data augmentation this drop can be further reduced and, for example in the $norm\_ipi10000\_AI3\_KD$

Pipeline	Dataset	Accuracy	TPR	TNR
norm_ipi10000_AI3_KD	KDD	84.39%	75.25%	96.46%
norm_ipi10000_AI3_KD	UNSW	77.05%	87.29%	71.14%
norm_ipi10000_RNA3_KD	KDD	82.42%	71.48%	96.87%
norm_ipi10000_CNA_KD	KDD	81.90%	70.65%	96.76%

Table 5.5: Evaluation results of the the three data augmentation modules after an Incremental Point Insertion module.

the classification result can be even improved by a few percentage points. However, this module also shows a problem: For the *CSE-CIC-IDS2018* dataset $9,764$ out of the $10,000$ instances are benign and thus the anomalies are underrepresented in this reduced datasets. This can be compensated by data augmentation or anomalies may not be needed for the k-distance algorithm.

5.5 Data Augmentation

Evaluating the data augmentation modules shows that these modules have a really bad performance. The evaluation time for each training set and pipeline would be longer than an hour. Therefore all three dataset were reduced to 10,000 instances with the Incremental Point Insertion module and the result is being stored. Afterwards, the reduced data sets were used for training the pipelines. Although this also takes a long time, the time effort is still reasonable.

All three modules show a slight increase of the accuracy in the *NSL-KDD* dataset and show also a high True Negative. In the other datasets this increase cannot be observed.

5.6 Filtered Anomalies

An important requirement for this thesis is that it is not necessary to label recorded data. The assumption when collecting data is that it only consists out of benign data. Thus, for all training sets data labeled anomalous where removed and then the classifier was trained. As it can be seen in Table 5.6, this method does not have a huge influence on the classification result. The result is a few percentage points worse, but with a transformation step and augmentation step it can be improved. For this, it is not important to have anomalous data in the training set. The pipeline with the transformation and augmentation for the *CSE-CIC-IDS2018* dataset is missing because the training and evaluation phase took more than twelve hours and was then manually aborted.

Pipeline	Dataset	Accuracy	TPR	TNR
FP_Norm_KD	KDD	76.13%	62.55%	94.08%
FP_Norm_YJT_RNA3_KD	KDD	84.27%	93.39%	72.21%
FP_Norm_KD	UNSW	72%	24.53%	99.4%
FP_Norm_YJT_RNA3_KD	UNSW	76.81%	40.93%	97.59%
FP_Norm_KD	IDS	93.7%	54.81%	99.37%

Table 5.6: Evaluation results when all anomalies were filtered out of the training sets.

Pipeline	Dataset	Accuracy	TPR	TNR
FP_norm_YJT_PCA15_RNA_KD	KDD	83.47%	91.60%	72.73%
	UNSW	79.82%	54.76%	94.28%
	IDS	95.34%	71.07%	98.88%
FP_norm_PCA15_YJT_RNA_KD	KDD	81.00%	72.64%	92.04%
	UNSW	75.17%	33.00%	99.51%
	IDS	93.78%	52.39%	99.82%
FP_norm_YJT_PCA15_KD	KDD	76.83%	77.68%	75.71%
	UNSW	69.73%	18.44%	99.33%
	IDS	95.59%	70.57%	99.24%

Table 5.7: Evaluation results of combined modules when all anomalies were filtered.

5.7 Combination of the best Modules

Modules like PCA or the Yeo-Johnson Transformer showed an improvement in the classification reports. So these modules are combined to one classifier. This results in several possible combinations, not all of this are evaluated because this would take to much time. For example, training the $FP\_Norm\_PCA5\_RNA1\_KD$ pipeline with the *NSL-KDD* dataset takes more than 2 hours, training the $FP\_Norm\_PCA5\_KD$ takes 3.3 seconds. Using a higher target dimension for the PCA would increase this relation even more. In Table 5.7 the pipelines with multiple modules are shown. It can be seen that a Yeo-Johnson transformation has a positive affect if it is before a PCA module. When it is after a PCA module it has a negative effect. Also the positive effect of the Random Noise module can be shown.

5.8 Performance of Modules

For measuring the performance of each module an AMD Ryzen 3900X with 32 GB DDR4-3600 RAM and a NVIDIA 2070 Super running on Windows 10 were used. The computation capabilities of the NVIDIA 2070 Super were only used for the neural network module and the autoencoder module. All other modules only used the CPU and only one core,

Module \ Batch size	1 Instance	10 Instances	100 Instances
Normalization	0.38s	0.07s	0.04s
PCA	1.18s	0.13s	0.02s
Isomap	2016.49s	219.21s	42.24s
Autoencoder	300.85s	29.36s	2.91s
Feature Removing	0.05s	0.01s	0.0s
Quantile Transformer	75.27s	7.55s	0.83s
Dependent Transformer	2.93s	0.3s	0.03s
Yeo-Johnson	83.93s	8.4s	0.92s
K-distance	23.58s	18.8s	18.77s
NN1	14.28s	1.49s	0.18s
NN5	29.24s	3.01s	0.32s

Table 5.8: The time in seconds each module needs to transform 22400 instances. Dimension reduction methods have a target dimension of 5.

because Python has a huge overhead for multiprocessing and its support for multithreading is so limited that is actually unusable in this application[7].

The test setup consists of the *NSL-KDD* training set and the *NSL-KDD* testing set. Each module is trained alone with the training set. After the training, the time that is needed to transform the testing set rounded down to the nearest hundred is measured. Dividing this time by the number of instances in the testing set gives the average time a module takes to transform one instance. Additionally, different batch sizes where tested. First, each instance is transformed independently (batch size = 1), then batches sized of ten and 100 are tested. Python has a huge overhead while running because it is a script language. By using batches some of it can be reduced. Moreover, the cache of the processor may be used more efficiently with batches.

In Table 5.8 the huge performance improvement can be seen. Most modules have a huge speedup, some even have a speedup factor of almost ten for the batch with a size of ten and almost a speedup factor of 100 for a batch size of 100. Only the k-distance module shows with 20.3% for batch size of ten a smaller improvement of the performance. Compared to this, almost no further improvement is achieved with a batch size of 100. All in all this shows that the instances should be collected in batches and should not be classified sequentially. On the one hand, this increases the performance, but on the other

[7] *"Getting multiple tasks running simultaneously requires a non-standard implementation of Python, writing some of your code in a different language, or using multiprocessing which comes with some extra overhead."* https://realpython.com/intro-to-python-threading/

hand also leads to an increased classification delay. However, this does not necessarily represent a problem. When the network traffic is low the batch size might be low as well and because of the low network traffic the performance is not as relevant. But when the network traffic is high, an increased batch size might compensate the computation cost. Therefore, this does not cause an unnecessary time delay.

Furthermore, Table 5.8 shows that the Isomap module should not be used. It has a terrible sequential performance and even with a batch size of 100 it is with 1.89ms per instance really slow. This means that about 50 instances can be handled per second when using a high batch size and assuming that the remaining modules in this pipeline are very fast.

During the evaluation another behavior was noticed. When using all cores for the k-distance module the CPU usage was 100% but neither the CPU temperature nor the CPU power consumption indicated such high a usage. But at the same time, the operating system and other programs felt sluggish. This might result from the fact that the memory controller limits the speed of this module.

5.9 Industrial Dataset in Comparison to Used Dataset

In context of this thesis also an own dataset was collected. It is from a real industrial facility and consists of five hours of real traffic. After it having be converted to flows with CICFlowMeter it consists of 36968 instances. The assumption of this work is that network traffic in an industrial context is simpler than in normal networks. However, it is not easy to measure its complexity because there is no heuristic that can answer it.

One method is to analyze the variance of the features. A dimension reduction to two dimensions by PCA can be used for this. The dimension reduction with PCA has the advantage that it is easy to plot and due to PCA's goal of reducing the embedding error this two dimensional embedding tells a lot about the characteristic of data. For all datasets used in this thesis this embedding is shown in Figure 5.1. It shows that the *CSE-CIC-IDS2018* dataset has a high data variance, but this dataset contains more instances than the other datasets. The *NSL-KDD* and *UNSW-NB15* datasets have a comparable size of instances, but the *NSL-KDD* shows a lot more variance in the data, but not as much as the *CSE-CIC-IDS2018* dataset. The own dataset shows a low variance in comparison to *UNSW-NB15* . Another interesting observation can be made regarding the time of capture that is represented by the color and was never used for training. Only the own dataset

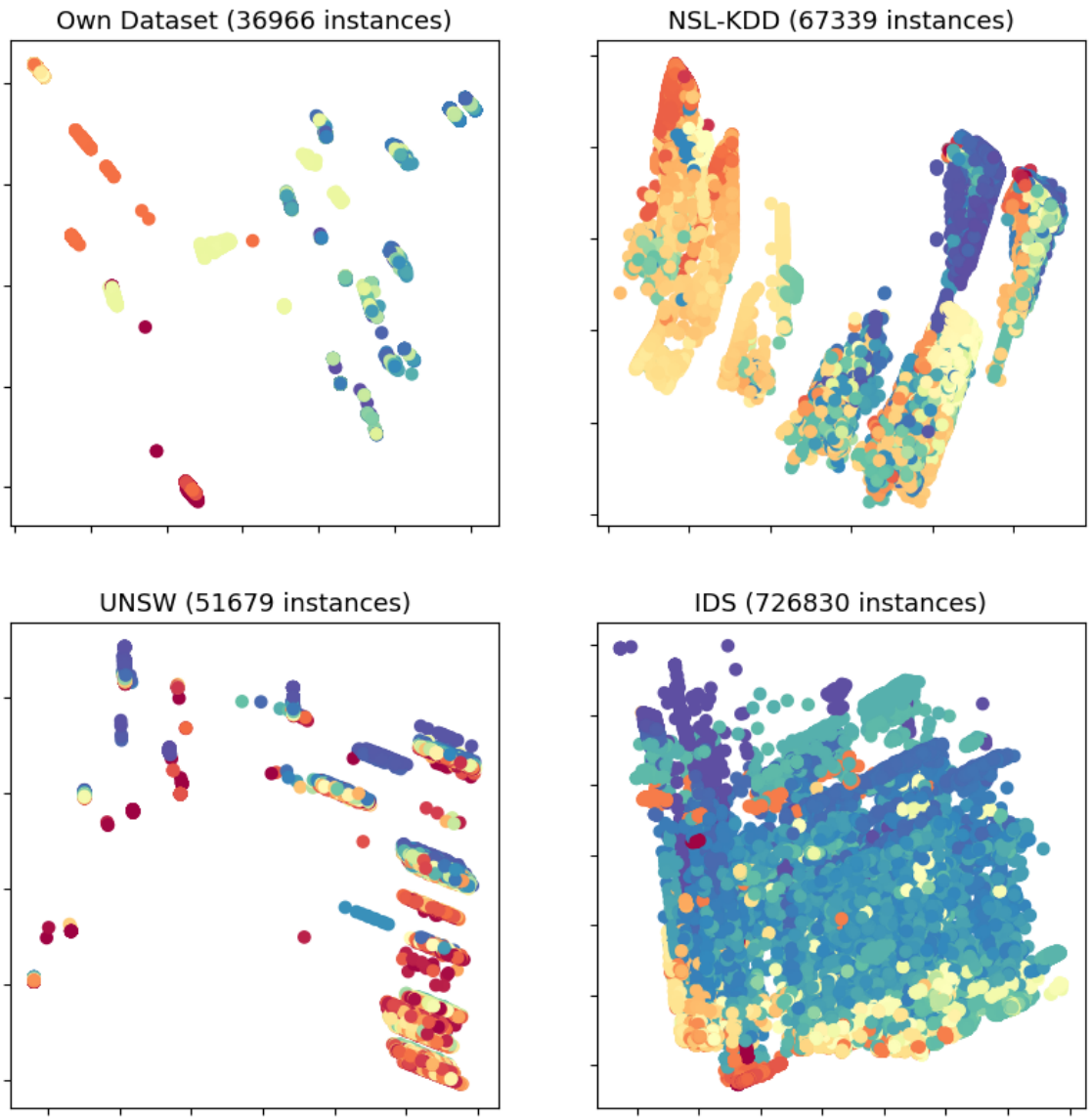

Figure 5.1: Each dataset represents the training set with only benign instances after reducing their dimension to two via PCA. The color represents the time of capture of this instance. The *CSE-CIC-IDS2018* dataset shows the most universal data and the own dataset the fewest. The instances are shuffled before drawing to prevent that instances that are captured later overlay early ones.

and the *CSE-CIC-IDS2018* dataset provided a timestamp when a specific instance was captured. In both datasets the instances are listed almost chronologically, only some are off by a few seconds, even less by some minutes. Therefore, the assumption is that this is also true for the other datasets which are thus colored, too. The observation is that

equally colored instances cluster together in all datasets. One can argue that this happens because these are only simulated datasets and at a different time different behavior was simulated. However, this is not true for the own dataset that consists of real network data. As regards the other datasets, their documentation does not mention it for benign data Anomalous data was not used for this figure.

The informative value of PCA in two dimension is limited, the low embeddings can have different embedding errors and the original number of features differs for each dataset. Therefore, quantiles are used to support the assumption that the characteristic of the own dataset has the lowest complexity. Computing the quantiles of the nearest neighbor distances shows that most of the instances of the own dataset lay in dense clusters. This is shown in Figure 5.2. Out of 36966 instances 36177 instances are included in the

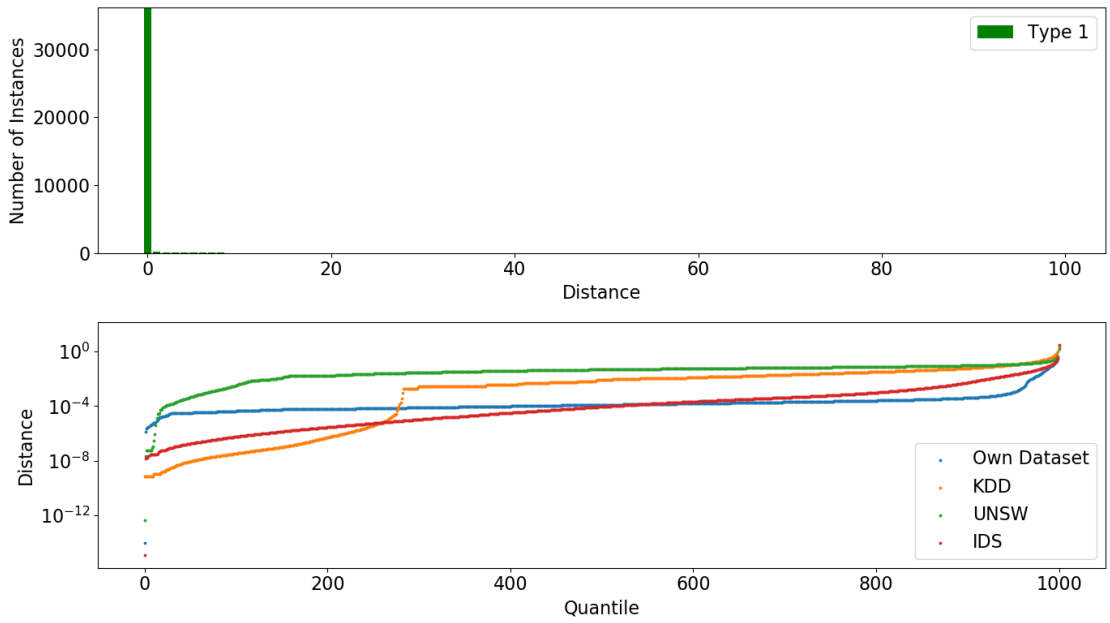

Figure 5.2: Above the count of instances in the first 100 out of 1000 quantiles of the maximum distance to the nearest neighbor in the own dataset computed like Figure 3.2. Below is the distance of the quantiles for each dataset.

first quantile. In relation to the other three dataset this is a much higher ratio, even if this dataset has the smallest amount of instances. Also, in the own dataset the distances are more homogeneous.

All in all, this method sustains the assumption that network traffic in industrial facilities is less complex than data traffic in general. Also, at one time there is less diversity in

the own dataset than in the other datasets.

6 Conclusions

6.1 Summary

The aim of the thesis was to show that with the help of anomaly detection an intrusion detection system can be designed and as a consequence of the anomaly detection only benign data are necessary for training the system. All in all it is possible to detect anomalies in network traffic and to declare them as attacks. Network traffic shows that it is possible to gain information on data packets without observing their payload. Similar packets cluster together and therefore it can be deducted where such a packet occur or if it is anomalous. Preprocessing the data has different effects. In some datasets one method is improving the classification result in another not. Good working modules chained together are not necessarily improving the result. It was shown that they might work worse than without any preprocessing.

Besides, anomalous data does not need to be present in the training data. With some algorithms it is still possible to distinguish them and with some preprocessing step they are not worse than with anomalies in the training set. But the computation time of the evaluation process limits the amount of complex pipelines.

6.2 Further Work

The classification overall makes some errors. These errors may be reduced be the frequency estimation method, but this still has to be evaluated. Also, this system does not learn over time. In the future there might be different approaches to enable this learning. Every time an alert is sent to a responsible person and this person declares this alert as a false positive, this data packets can be added to the training set as benign data. So over time the classifier will learn rare occurring behavior over time and the false alarm rate will decrease. A other way is that the classifier decides by itself which instances can be added to the dataset. This might be done by the scores. If an instance occurs within a range of a specific score, maybe between 0.1 and 0.2, this means that the classifier is relatively sure that this instance is benign, but it still have a slight doubt. If all other instances of this data stream have a low score, it can be sure that these instances are all benign and thus they can be added to the training set.

An approach to achieve a better classification might be the use of a two dimensional pipeline. Here, the modules are not included within one single sequence but in a few and these sequences can be connected between specific modules. For example, sequence a gets all features as an input, sequence b all continuous and sequence c all binary features. Sequence a has as only one module, a neural network. The first module of sequence b is PCA, the first one of sequence c is Isomap. In b the k-distance module then uses the input of the PCA and Isomap module, c uses a neural network with the input from Isomap. At the end, a majority vote determines the final score.

Glossary

TP True-positive

FP False-positive

TN True-negative

FN False-negative

FAR False acceptance rate

TPR True-positive rate. Defined as:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

TNR True-negative rate. Defined as:

$$\text{TNR} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

Precision Metric for the accuracy of a prediction. Defined as:

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

Recall Metric for the accuracy of a prediction. Defined as:

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

F1 Score Metric for an evaluation of a prediction. Defined as harmonic mean between *precision* and *recall*.

$$\text{F1} = \frac{2 \cdot \text{TP}}{2 \cdot \text{TP} + \text{FP} + \text{FN}}$$

Categorical cross entropy Loss function for problems where only one result is correct. Defined as:

$$L(y, \hat{y}) = -\frac{1}{n} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij} \cdot \log(\hat{y}_{ij})$$

where y is the one hot vector of the label, $\hat{y}$ is the predicted value, M is the number of samples and N is the number of classes.

Appendix

1 Influence of Features

Feature	Loss	Feature	Loss
duration	0.07799129	udp	0.07825942
tcp	0.07417392	icmp	0.07704299
other	0.08729411	private	0.11195250
http	0.08118024	remote_job	0.08145064
ftp_data	0.07763332	name	0.08553506
netbios_ns	0.07977720	eco_i	0.08148290
mtp	0.08341575	telnet	0.08114966
finger	0.07781831	domain_u	0.09576516
supdup	0.08059184	uucp_path	0.08324952
Z39_50	0.08507044	smtp	0.08748472
csnet_ns	0.08231732	uucp	0.08573607
netbios_dgm	0.08143534	urp_i	0.10889248
auth	0.08133500	domain	0.07516136
ftp	0.07408004	bgp	0.08121736
ldap	0.08208767	ecr_i	0.09442041
gopher	0.09497014	vmnet	0.08177768
systat	0.08190961	http_443	0.08608453
efs	0.08363424	whois	0.08556235
imap4	0.08495196	iso_tsap	0.08217800
echo	0.08195561	klogin	0.08198339
link	0.08523845	sunrpc	0.07828239
login	0.08420789	kshell 2	0.08196889
sql_net	0.08133991	time	0.07424584
hostnames	0.08227906	exec	0.07909843
ntp_u	0.09384349	discard	0.08165737
nntp	0.08631565	courier	0.08493748
ctf	0.08484051	ssh	0.07834175
daytime	0.08100215	shell u	0.07342729
netstat	0.08145552	pop_3	0.08843989

Feature	Loss	Feature	Loss
nnsp	0.08406656	IRC	0.13469801
pop_2	0.07743456	printer	0.07598141
tim_i	0.07397961	pm_dump	0.08048942
red_i	0.07613141	netbios_ssn	0.08039548
rje	0.08298462	X11	0.07783800
urh_i	0.07678504	http_8001	0.07416794
aol	0.07417775	http_2784	0.07389954
tftp_u	0.07530086	harvest	0.07355261
SF	0.07707235	S0	0.08256348
REJ	0.08865889	RSTR	0.10281301
SH	0.07382112	RSTO	0.07563250
S1	0.10722392	RSTO0	0.08777098
S3	0.07726289	S2	0.07776331
OTH	0.07469919	src_bytes	0.08868196
dst_bytes	0.08402774	land	0.07647929
wrong_fragment	0.18789059	urgent	0.07475666
hot	0.24220038	num_failed_logins	0.08606800
logged_in	0.07594330	num_compromised	0.07698736
root_shell	0.08061794	su_attempted	0.08344443
num_root	0.08474115	num_file_creations	0.07521413
num_shells	0.07420022	num_access_files	0.07960459
num_outbound_cmds	0.07392417	is_host_login	0.07415355
is_guest_login	0.07546003	count	0.10925893
srv_count	0.08437745	serror_rate	0.0754859
srv_serror_rate	0.08139401	rerror_rate	0.07679147
srv_rerror_rate	0.08775885	same_srv_rate	0.08267055
diff_srv_rate	0.07709735	srv_diff_host_rate	0.07858630
dst_host_count	0.07771403	dst_host_srv_count	0.10498699
dst_host_same_srv_rate	0.07646997	dst_host_diff_srv_rate	0.07668056
dst_host_same_src_port_rate	0.09975468	dst_host_srv_diff_host_rate	0.08572867
dst_host_serror_rate	0.07696668	dst_host_srv_serror_rate	0.08132237
dst_host_rerror_rate	0.11900928	dst_host_srv_rerror_rate	0.07534955
default	0.07252713		

Table 2: Loss in the kdd dataset for individual features

Feature	Loss	Feature	Loss
duration	0.16687561	tcp	0.19770894
udp	0.18889586	arp	0.37075877
ospf	0.16669606	icmp	0.19147224
igmp	0.33376026	rtp	0.16352813
ddp	0.19197491	ipv6-frag	0.19919494
cftp	0.19751718	wsn	0.20588426
pvp	0.19919437	wb-expak	0.19540352
mtp	0.19794150	pri-enc	0.19061939
sat-mon	0.20099395	cphb	0.20237636
sun-nd	0.25026256	iso-ip	0.19891470
xtp	0.19735988	il	0.20147826
unas	0.26499309	mfe-nsp	0.19650693
3pc	0.19709803	ipv6-route	0.19950248
idrp	0.19870882	bna	0.19888122
swipe	0.25036776	kryptolan	0.19894208
cpnx	0.20605726	rsvp	0.21401180
wb-mon	0.19532411	vmtp	0.19708201
ib	0.19368166	dgp	0.19745453
eigrp	0.20250455	ax.25	0.19672597
gmtp	0.19184809	pnni	0.19349478
sep	0.20521484	pgm	0.19327939
idpr-cmtp	0.19345385	zero	0.18699829
rvd	0.19296825	mobile	0.24697231
narp	0.19706660	fc	0.20014873
pipe	0.20541117	ipcomp	0.19936251
ipv6-no	0.19806418	sat-expak	0.19616569
ipv6-opts	0.19370064	snp	0.19748441
ipcv	0.20484861	br-sat-mon	0.20196981
ttp	0.20093680	tcf	0.19951362
nsfnet-igp	0.19854100	sprite-rpc	0.20243152
aes-sp3-d	0.19688622	sccopmce	0.20042587
sctp	0.40032061	qnx	0.19772061
scps	0.19448230	etherip	0.19263548
aris	0.19348525	pim	0.20513516
compaq-peer	0.19751139	vrrp	0.19824971
iatp	0.18698589	stp	0.19272366

Feature	Loss	Feature	Loss
l2tp	0.19073143	srp	0.19195287
sm	0.19312451	isis	0.19360433
smp	0.19603953	fire	0.19569305
ptp	0.19194979	crtp	0.20498675
sps	0.20010221	merit-inp	0.19519254
idpr	0.19376590	skip	0.19665836
any	0.21764762	larp	0.19726016
ipip	0.19440112	micp	0.19891700
encap	0.18737500	ifmp	0.19099978
tp++	0.19728625	a/n	0.19487210
ipv6	0.20381330	i-nlsp	0.19819670
ipx-n-ip	0.19706770	sdrp	0.19470979
tlsp	0.19987414	gre	0.20592967
mhrp	0.19556622	ddx	0.19058098
ippc	0.20398189	visa	0.20268971
secure-vmtp	0.19886084	uti	0.19540804
vines	0.19893944	crudp	0.19865839
iplt	0.19931163	ggp	0.18985730
ip	0.18429032	ipnip	0.18440227
st2	0.18252852	argus	0.19353995
bbn-rcc	0.19302267	egp	0.18728637
emcon	0.19486464	igp	0.18838040
nvp	0.18968685	pup	0.18779816
xnet	0.20187326	chaos	0.20280351
mux	0.20122743	dcn	0.20463413
hmp	0.20407148	prm	0.20308074
trunk-1	0.20056311	xns-idp	0.20292466
leaf-1	0.20174025	leaf-2	0.20381040
rdp	0.20316632	irtp	0.19781246
iso-tp4	0.20112035	netblt	0.20371135
trunk-2	0.20320567	cbt	0.18486751
-	0.18049189	ftp	0.18477398
smtp	0.19205237	snmp	0.15924455
http	0.15367991	ftp-data	0.16013889
dns	0.20273510	ssh	0.19412258
radius	0.15877332	pop3	0.23052034

Feature	Loss	Feature	Loss
dhcp	0.23411682	ssl	0.19213782
irc	0.18215303	FIN	0.15591057
INT	0.18581892	CON	0.31756878
ECO	0.16788009	REQ	0.23414861
RST	0.16786147	PAR	0.15266710
URN	0.15654806	no	0.15395254
spkts	0.20267089	dpkts	0.16153087
sbytes	0.32996132	dbytes	0.21577246
rate	0.15065439	sttl	0.22923831
dttl	0.66424425	sload	0.15545375
dload	0.73463322	sloss	0.40741691
dloss	0.18858795	sinpkt	0.16456611
dinpkt	0.16297543	sjit	0.15138790
djit	0.16774700	swin	0.19949995
stcpb	0.14813813	dtcpb	0.14808381
dwin	0.16473744	tcprtt	0.21997018
synack	0.48150873	ackdat	0.16166112
smean	0.15019096	dmean	0.30004921
trans_depth	0.32024338	response_body_len	0.27530478
ct_srv_src	0.24777547	ct_state_ttl	0.25534593
ct_dst_ltm	0.17229248	ct_src_dport_ltm	0.20121059
ct_dst_sport_ltm	0.20305612	ct_dst_src_ltm	0.20152032
is_ftp_login	0.28222330	ct_ftp_cmd	0.27928564
ct_flw_http_mthd	0.17003394	ct_src_ltm	0.15226068
ct_srv_dst	0.48738252	is_sm_ips_ports	0.36087954
default	0.14793465		

Table 4: Loss in the UNSW dataset for individual features

Feature	Loss	Feature	Loss
Dst Port	0.04968043	Protocol	0.02449291
Flow Duration	0.12805765	Tot Fwd Pkts	1.60549708
Tot Bwd Pkts	0.11893603	TotLen Fwd Pkts	0.12567758
TotLen Bwd Pkts	0.16908513	Fwd Pkt Len Max	0.19879829
Fwd Pkt Len Min	0.16866534	Fwd Pkt Len Mean	0.10053958
Fwd Pkt Len Std	0.05418228	Bwd Pkt Len Max	0.05747679
Bwd Pkt Len Min	0.24410540	Bwd Pkt Len Mean	0.11714780
Bwd Pkt Len Std	0.06157014	Flow Byts/s	0.18509712
Flow Pkts/s	0.08688144	Flow IAT Mean	0.15837885
Flow IAT Std	0.18591565	Flow IAT Max	0.48137715
Flow IAT Min	0.07041742	Fwd IAT Tot	0.11299740
Fwd IAT Mean	0.11132844	Fwd IAT Std	0.21538602
Fwd IAT Max	0.28731462	Fwd IAT Min	0.08891705
Bwd IAT Tot	0.32674475	Bwd IAT Mean	0.15326110
Bwd IAT Std	0.14621802	Bwd IAT Max	0.13282232
Bwd IAT Min	0.24074095	Fwd PSH Flags	0.03360106
Bwd PSH Flags	0.02442084	Fwd URG Flags	0.03858201
Bwd URG Flags	0.02778878	Fwd Header Len	3.17210150
Bwd Header Len	0.25039605	Fwd Pkts/s	0.17548326
Bwd Pkts/s	0.13981359	Pkt Len Min	0.08656507
Pkt Len Max	0.04419004	Pkt Len Mean	0.19253987
Pkt Len Std	0.03091833	Pkt Len Var	0.15996076
FIN Flag Cnt	0.05311864	SYN Flag Cnt	0.02687676
RST Flag Cnt	0.11372499	PSH Flag Cnt	0.02835967
ACK Flag Cnt	0.05755865	URG Flag Cnt	0.04180483
CWE Flag Count	0.04360356	ECE Flag Cnt	0.10522234
Down/Up Ratio	1.60358700	Pkt Size Avg	0.13934684
Fwd Seg Size Avg	0.10786053	Bwd Seg Size Avg	0.11123708
Fwd Byts/b Avg	0.02462303	Fwd Pkts/b Avg	0.02529462
Fwd Blk Rate Avg	0.02666364	Bwd Byts/b Avg	0.02949348
Bwd Pkts/b Avg	0.02439664	Bwd Blk Rate Avg	0.02452544
Subflow Fwd Pkts	1.53210364	Subflow Fwd Byts	0.12148861
Subflow Bwd Pkts	0.13198444	Subflow Bwd Byts	0.16670219
Init Fwd Win Byts	0.04447886	Init Bwd Win Byts	0.37370590
Fwd Act Data Pkts	1.43621841	Fwd Seg Size Min	0.08795218
Active Mean	0.33446873	Active Std	0.29263375

Feature	Loss	Feature	Loss
Active Max	0.36178419	Active Min	0.27079734
Idle Mean	0.09875786	Idle Std	0.10353051
Idle Max	0.10929057	Idle Min	0.11061051
default	0.02135365		

Table 6: Loss in the ids2018 dataset for individual features

2 Definition of Neural Networks

NN1:

Layer Type	Units	Activation Type
Fully Connected	10	Relu
Fully Connected	2	Softmax

NN2:

Layer Type	Units	Activation Type
Fully Connected	40	Relu
Fully Connected	20	Relu
Fully Connected	2	Softmax

NN3:

Layer Type	Units	Activation Type
Fully Connected	80	Relu
Fully Connected	40	Relu
Fully Connected	2	Softmax

NN4:

Layer Type	Units	Activation Type
Fully Connected	80	Relu
Fully Connected	60	Relu
Fully Connected	30	Relu
Fully Connected	2	Softmax

NN5:

Layer Type	Units	Activation Type
Fully Connected	100	Relu
Fully Connected	80	Relu
Fully Connected	60	Relu
Fully Connected	40	Relu
Fully Connected	2	Softmax

3 Evaluation Results

Name	Dataset	Accuracy	TPR	TNR
norm_NN1	KDD	0.795517	0.662266	0.971609
	UNSW	0.747062	0.980699	0.612232
	IDS	0.972469	0.80267	0.997244
norm_NN2	KDD	0.804634	0.700703	0.941978
	UNSW	0.751307	0.984802	0.616558
	IDS	0.974728	0.809627	0.998818
norm_NN3	KDD	0.816553	0.729922	0.931034
	UNSW	0.770085	0.988804	0.643864
	IDS	0.976271	0.83177	0.997354
norm_NN4	KDD	0.789247	0.688594	0.922259
	UNSW	0.818337	0.967477	0.732269
	IDS	0.976901	0.831516	0.998113
norm_NN5	KDD	0.80579	0.713984	0.927111
	UNSW	0.806566	0.958055	0.719143
	IDS	0.977214	0.830636	0.9986
norm_pca5_NN1	KDD	0.778529	0.635312	0.967789
	UNSW	0.702591	0.905572	0.585453
	IDS	0.891517	0.244789	0.985878
norm_pca5_NN2	KDD	0.759851	0.629453	0.93217
	UNSW	0.735569	0.995187	0.585745
	IDS	0.916073	0.376116	0.994856
norm_pca5_NN3	KDD	0.763898	0.643125	0.923498
	UNSW	0.736514	0.99848	0.585336
	IDS	0.964886	0.726242	0.999706
norm_pca5_NN4	KDD	0.748599	0.619687	0.918955
	UNSW	0.740537	0.990324	0.596387
	IDS	0.965461	0.733769	0.999267
norm_pca5_NN5	KDD	0.76203	0.645078	0.916581
	UNSW	0.723538	0.997822	0.565252
	IDS	0.965078	0.734659	0.998697
norm_FR5_NN1	KDD	0.669572	0.49125	0.905224
	UNSW	0.682015	0.984347	0.507543
	IDS	0.927069	0.500771	0.989269
	KDD	0.691452	0.530625	0.903985

Name	Dataset	Accuracy	TPR	TNR
norm_FR5_NN2	UNSW	0.684388	0.992806	0.506402
	IDS	0.929495	0.523411	0.988745
	KDD	0.708574	0.561797	0.90254
norm_FR5_NN3	UNSW	0.684351	0.992705	0.506402
	IDS	0.944468	0.667413	0.984892
	KDD	0.689585	0.525	0.907082
norm_FR5_NN4	UNSW	0.685371	0.992503	0.508127
	IDS	0.944412	0.660138	0.985889
	KDD	0.682647	0.511797	0.908425
norm_FR5_NN5	UNSW	0.693935	0.989058	0.523622
	IDS	0.92208	0.732498	0.949741
	KDD	0.69421	0.504219	0.945282
norm_auto5_NN1	UNSW	0.581174	0.712614	0.505321
	IDS	0.834162	0.031025	0.951345
	KDD	0.606377	0.343125	0.954264
norm_auto5_NN2	UNSW	0.615097	0.900101	0.450623
	IDS	0.915909	0.369761	0.995595
	KDD	0.806457	0.698047	0.949721
norm_auto5_NN3	UNSW	0.695084	0.872442	0.592732
	IDS	0.936624	0.560704	0.991473
	KDD	0.767589	0.698984	0.858249
norm_auto5_NN4	UNSW	0.620213	0.863425	0.479857
	IDS	0.895666	0.189454	0.998707
	KDD	0.818821	0.746719	0.914103
norm_auto5_NN5	UNSW	0.675398	0.825785	0.58861
	IDS	0.960831	0.72389	0.995402
	KDD	0.751579	0.618516	0.927421
norm_ipi10000_NN1	UNSW	0.721129	0.990932	0.565427
	IDS	0.872704	0.000251	1.0
	KDD	0.77208	0.654375	0.927628
norm_ipi10000_NN2	UNSW	0.737311	0.991489	0.590627
	IDS	0.891669	0.15209	0.999578
	KDD	0.765054	0.641172	0.928763
norm_ipi10000_NN3	UNSW	0.738442	0.997163	0.589136
	IDS	0.901417	0.229444	0.999462
	KDD	0.784488	0.659531	0.949618

Name	Dataset	Accuracy	TPR	TNR
norm_ipi10000_NN4	UNSW	0.749231	0.98536	0.612963
	IDS	0.899749	0.217724	0.999261
	KDD	0.792849	0.665	0.961801
norm_ipi10000_NN5	UNSW	0.740092	0.988045	0.597001
	IDS	0.945989	0.759637	0.973179
	KDD	0.520457	0.674922	0.316333
norm_QTT_NN1	UNSW	0.740537	0.984549	0.599719
	IDS	0.971267	0.795163	0.996961
	KDD	0.616695	0.662813	0.555751
norm_QTT_NN2	UNSW	0.766025	0.96155	0.653189
	IDS	0.977523	0.840054	0.99758
	KDD	0.780664	0.651797	0.95096
norm_QTT_NN3	UNSW	0.78332	0.950456	0.686868
	IDS	0.978006	0.837958	0.99844
	KDD	0.750289	0.657813	0.872496
norm_QTT_NN4	UNSW	0.754699	0.970871	0.629948
	IDS	0.97582	0.827851	0.99741
	KDD	0.610113	0.675	0.524365
norm_QTT_NN5	UNSW	0.767768	0.963323	0.654914
	IDS	0.97823	0.837346	0.998785
	KDD	0.803656	0.685547	0.959736
norm_DST_NN1	UNSW	0.752864	0.982219	0.620505
	IDS	0.972626	0.813677	0.995817
	KDD	0.806947	0.717656	0.924943
norm_DST_NN2	UNSW	0.764246	0.987234	0.635561
	IDS	0.977309	0.830103	0.998787
	KDD	0.785956	0.678984	0.927318
norm_DST_NN3	UNSW	0.774905	0.976342	0.658656
	IDS	0.976736	0.839748	0.996723
	KDD	0.794139	0.698516	0.920504
norm_DST_NN4	UNSW	0.801413	0.981104	0.697714
	IDS	0.976246	0.834391	0.996943
	KDD	0.800187	0.705859	0.92484
norm_DST_NN5	UNSW	0.782041	0.973708	0.671432
	IDS	0.976791	0.833198	0.997742
	KDD	0.798897	0.707031	0.920297

Name	Dataset	Accuracy	TPR	TNR
norm_YJS_NN1	UNSW	0.775183	0.978977	0.657575
	IDS	0.974718	0.821415	0.997085
	KDD	0.797341	0.702109	0.923188
norm_YJS_NN2	UNSW	0.816205	0.969858	0.727533
	IDS	0.977913	0.837644	0.998379
	KDD	0.805168	0.722187	0.914826
norm_YJS_NN3	UNSW	0.81424	0.957447	0.731597
	IDS	0.97814	0.841785	0.998035
	KDD	0.826292	0.726953	0.957568
norm_YJS_NN4	UNSW	0.816854	0.963222	0.732386
	IDS	0.97796	0.838717	0.998277
	KDD	0.791337	0.657891	0.967685
norm_YJS_NN5	UNSW	0.779372	0.967882	0.670584
	IDS	0.978343	0.838004	0.998819
	KDD	0.794984	0.823984	0.756659
norm_ipi10000_PCA5_AI_NN1	UNSW	0.520502	1.0	0.243788
	IDS	0.574454	0.334648	0.609444
	KDD	0.760384	0.6	0.972331
norm_ipi10000_PCA5_AI_NN2	UNSW	0.599933	0.999848	0.369146
	IDS	0.714529	0.048218	0.811748
	KDD	0.78769	0.645859	0.975119
norm_ipi10000_PCA5_AI_NN3	UNSW	0.612186	0.999696	0.388558
	IDS	0.752092	0.015877	0.85951
	KDD	0.799831	0.667891	0.97419
norm_ipi10000_PCA5_AI_NN4	UNSW	0.716365	0.999645	0.552885
	IDS	0.808424	0.020833	0.923338
	KDD	0.792182	0.665312	0.959839
norm_ipi10000_PCA5_AI_NN5	UNSW	0.622511	0.999696	0.404841
	IDS	0.80614	0.016759	0.921316
	KDD	0.733568	0.545078	0.982655
norm_ipi10000_PCA5_RNA_NN1	UNSW	0.602102	0.937183	0.408729
	IDS	0.872672	0.0	1.0
	KDD	0.723606	0.526797	0.983688
norm_ipi10000_PCA5_RNA_NN2	UNSW	0.714826	0.997163	0.551891
	IDS	0.872672	0.0	1.0
	KDD	0.725029	0.528125	0.985236

Name	Dataset	Accuracy	TPR	TNR
norm_ipi10000_PCA5_RNA_NN3	UNSW	0.728784	0.996707	0.574168
	IDS	0.872672	0.0	1.0
norm_ipi10000_PCA5_RNA_NN4	KDD	0.72872	0.537109	0.981933
	UNSW	0.730026	0.997872	0.575455
	IDS	0.872699	0.000214	1.0
norm_ipi10000_PCA5_RNA_NN5	KDD	0.772169	0.614531	0.980487
	UNSW	0.730972	0.997872	0.576946
	IDS	0.873442	0.006044	1.0
norm_ipi10000_PCA5_CN_NN1	KDD	0.737036	0.553281	0.979868
	UNSW	0.641419	0.985714	0.442729
	IDS	0.872672	0.0	1.0
norm_ipi10000_PCA5_CN_NN2	KDD	0.772703	0.620156	0.974293
	UNSW	0.737256	0.991185	0.590715
	IDS	0.872672	0.0	1.0
norm_ipi10000_PCA5_CN_NN3	KDD	0.760473	0.600234	0.972228
	UNSW	0.737552	0.99382	0.589663
	IDS	0.873616	0.007417	1.0
norm_ipi10000_PCA5_CN_NN4	KDD	0.788224	0.654531	0.964898
	UNSW	0.735847	0.999341	0.583786
	IDS	0.865196	0.009192	0.990092
norm_ipi10000_PCA5_CN_NN5	KDD	0.745219	0.561875	0.987508
	UNSW	0.729619	0.999645	0.573788
	IDS	0.873975	0.010266	0.999995
norm_KD	KDD	0.794984	0.693672	0.928866
	UNSW	0.788418	0.908308	0.719231
	IDS	0.9633	0.872323	0.976574
norm_PCA5_KD	KDD	0.840479	0.781406	0.918542
	UNSW	0.776795	0.886879	0.713267
	IDS	0.964278	0.868573	0.978241
norm_FR5_KD	KDD	0.677533	0.507734	0.90192
	UNSW	0.70819	0.842958	0.630416
	IDS	0.955258	0.855447	0.969821
norm_AE5_KD	KDD	0.793294	0.690469	0.929176
	UNSW	0.637137	0.705674	0.597585
	IDS	0.957895	0.834112	0.975956
	KDD	0.658143	0.710781	0.588581

Name	Dataset	Accuracy	TPR	TNR
norm_QT_KD	UNSW	0.777314	0.913273	0.698854
	IDS	0.961368	0.878122	0.973514
norm_DS_KD	KDD	0.803745	0.709531	0.928247
	UNSW	0.800968	0.93845	0.721628
	IDS	0.962365	0.873186	0.975376
norm_YJ_KD	KDD	0.845459	0.789062	0.919988
	UNSW	0.796037	0.925937	0.721072
	IDS	0.960732	0.875885	0.973111
norm_ipi10000_KD	KDD	0.783198	0.679688	0.919988
	UNSW	0.774719	0.902533	0.700959
	IDS	0.943404	0.769061	0.968842
norm_ipi10000_AI3_KD	KDD	0.843947	0.752578	0.964691
	UNSW	0.77053	0.872948	0.711425
	IDS	0.930993	0.736109	0.959428
norm_ipi10000_RNA3_KD	KDD	0.824202	0.714844	0.968718
	UNSW	0.771179	0.872391	0.71277
	IDS	0.93135	0.735819	0.959879
norm_ipi10000_CNA_KD	KDD	0.819043	0.706562	0.967685
	UNSW	0.768287	0.87386	0.707361
	IDS	0.931161	0.736057	0.959628
norm_ipi10000_ISO_NN1	KDD	0.72285	0.518828	0.992463
	UNSW	0.535054	0.838754	0.359791
	IDS	0.872672	0.0	1.0
norm_ipi10000_ISO_NN2	KDD	0.776528	0.652344	0.940636
	UNSW	0.634357	0.983435	0.432907
	IDS	0.872672	0.0	1.0
norm_ipi10000_ISO_NN3	KDD	0.757271	0.5875	0.981623
	UNSW	0.630482	0.986272	0.425159
	IDS	0.872672	0.0	1.0
norm_ipi10000_ISO_NN4	KDD	0.749355	0.632812	0.903366
	UNSW	0.642216	0.984904	0.444454
	IDS	0.872672	0.0	1.0
norm_ipi10000_ISO_NN5	KDD	0.764298	0.631016	0.940429
	UNSW	0.643848	0.995593	0.440858
	IDS	0.872672	0.0	1.0
	KDD	0.774126	0.668438	0.913793

Name	Dataset	Accuracy	TPR	TNR
norm_ipi10000_ISO_KD	UNSW	0.756275	0.872594	0.689148
	IDS	0.935696	0.767939	0.960172
FP_norm_PCA5_YJT_RNA_KD	KDD	0.657698	0.421953	0.969234
	UNSW	0.710785	0.214843	0.996989
	IDS	0.872913	0.00765	0.99916
FP_norm_YJT_PCA5_RNA_KD	KDD	0.722716	0.526172	0.982449
	UNSW	0.730045	0.273404	0.993568
	IDS	0.87366	0.021999	0.997923
FP_norm_YJT_PCA5_YJT_RNA_KD	KDD	0.71778	0.515781	0.98472
	UNSW	0.680866	0.140628	0.992633
	IDS	0.874317	0.024044	0.998376
FP_norm_PCA15_YJT_RNA_KD	KDD	0.810015	0.726484	0.920401
	UNSW	0.751789	0.330091	0.995147
	IDS	0.937893	0.523915	0.998295
FP_norm_YJT_PCA15_RNA_KD	KDD	0.834742	0.916016	0.727338
	UNSW	0.798261	0.54767	0.942876
	IDS	0.953422	0.710721	0.988834
FP_norm_YJT_PCA15_YJT_RNA_KD	KDD	0.837588	0.924531	0.722693
	UNSW	0.788288	0.532573	0.935859
	IDS	0.9381	0.573219	0.991338
FP_norm_PCA5_YJT_KD	KDD	0.655074	0.420391	0.965208
	UNSW	0.737979	0.296353	0.992838
	IDS	0.9201	0.405557	0.995174
FP_norm_YJT_PCA5_KD	KDD	0.728142	0.538516	0.978732
	UNSW	0.738238	0.29767	0.992487
	IDS	0.894165	0.210922	0.993854
FP_norm_YJT_PCA5_YJT_KD	KDD	0.742151	0.561797	0.980487
	UNSW	0.688615	0.163576	0.99161
	IDS	0.880787	0.106027	0.993829
FP_norm_PCA15_YJT_KD	KDD	0.712488	0.543281	0.936093
	UNSW	0.735495	0.284144	0.995966
	IDS	0.936438	0.536677	0.994765
FP_norm_YJT_PCA15_KD	KDD	0.768389	0.776875	0.757175
	UNSW	0.697364	0.184448	0.993364
	IDS	0.955955	0.705752	0.992461
	KDD	0.749489	0.741797	0.759653

Name	Dataset	Accuracy	TPR	TNR
FP_norm_YJT_PCA15_YJT_KD	UNSW	0.693582	0.174823	0.992954
	IDS	0.947321	0.635182	0.992864

References

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[BC] G. E. P. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)*, 26(2):211–243.

[blo] Building autoencodres in Keras: https://blog.keras.io/building-autoencoders-in-keras.html.

[C+15] François Chollet et al. Keras. `https://keras.io`, 2015.

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41, 07 2009.

[CSE] A Realistic Cyber Defense Dataset: https://registry.opendata.aws/cse-cic-ids2018/.

[CYJX17] Yin Chuanlong, Zhu Yuefei, Fei Jinlong, and He Xinzheng. A deep learning approach for intrusion detection using recurrent neural networks. *IEEE Access*, 10 2017.

[DLOM02] Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. Frequency estimation of internet packet streams with limited space. In Rolf Möhring and Rajeev Raman, editors, *Algorithms — ESA 2002*, pages 348–360, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[FBF77] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, September 1977.

[Feaa] DERIVED FEATURES: http://kdd.ics.uci.edu/databases/kddcup99/task.html.

[Feab] UNSW-NB15 attributes: https://link.springer.com/article/10.1007/s00521-019-04396-2/tables/17.

[Feac] List of extracted traffic features: https://www.unb.ca/cic/datasets/ids-2018.html.

[GLMG16] Gerard Drapper Gil, Arash Habibi Lashkari, Mohammad Mamun, and Ali A. Ghorbani. Characterization of encrypted and vpn traffic using time-related features. 2016.

[JNSA15] Ahmad Y. Javaid, Quamar Niyaz, Weiqing Sun, and Mansoor Alam. A deep learning approach for network intrusion detection system. In *ICST Trans. Security Safety*, 2015.

[KDD] KDD Cup 1999: http://kdd.ics.uci.edu/databases/kddcup99/task.html.

[LDGMG17] Arash Habibi Lashkari, Gerard Draper-Gil, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. Characterization of tor traffic using time based features. 2017.

[Lil67] Hubert W. Lilliefors. On the kolmogorov-smirnov test for normality with mean and variance unknown. 1967.

[LYX19] Peng Lin, Kejiang Ye, and Cheng-Zhong Xu. Dynamic network anomaly detection system by using deep learning techniques. In Dilma Da Silva, Qingyang Wang, and Liang-Jie Zhang, editors, *Cloud Computing – CLOUD 2019*, pages 161–176, Cham, 2019. Springer International Publishing.

[MCS18] Nour Moustafa, Gideon Creech, and Jill Slay. Anomaly detection system using beta mixture models and outlier detection. In Prasant Kumar Pattnaik, Siddharth Swarup Rautaray, Himansu Das, and Janmenjoy Nayak, editors, *Progress in Computing, Analytics and Networking*, pages 125–135, Singapore, 2018. Springer Singapore.

[MHS18] Nour Moustafa, Jiankun Hu, and Jill Slay. A holistic review of network anomaly detection systems: A comprehensive survey. *Journal of Network and Computer Applications*, 128, 12 2018.

[MS15] N. Moustafa and J. Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 Military Communications and Information Systems Conference (MilCIS)*, pages 1–6, 2015.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[skla] Comparison of Manifold Learning methods: https://scikit-learn.org/stable/auto_examples/manifold/plot_compare_methods.html.

[sklb] Scikit-learn implemantation of the isomap algorithm: https://github.com/scikit-learn/scikit-learn/blob/fd237278e/sklearn/manifold/_isomap.py.

[TBLG] Mahbod Tavallae, Ebrahim Baghe, Wei Lu, and Ali A. Ghorbani. A detailed analysis of the kdd cup 99 data set. *CISDA 2009*.

[UNB] NSL-KDD: https://www.unb.ca/cic/datasets/nsl.html.

[WKR18] C. Wressnegger, A. Kellner, and K. Rieck. Zoe: Content-based anomaly detection for industrial control systems. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 127–138, June 2018.

[YJ00] In-Kwon Yeo and Richard A. Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959, 12 2000.