Marshall Plan Scholarship Report

Reduction of False Positives in Smart Grid Intrusion Detection

Bowling Green State University (BGSU), Ohio, USA
Salzburg University of Applied Sciences (SUAS), Salzburg, Austria

Submmitted by:

Joris Lückenga, BSc

BGSU: Prof. Robert R. Green II, Ph.D

SUAS: FH-Prof. DI Mag. Dr. Dominik Engel

By order of the

Austrian Marshall Plan Foundation

Salzburg, September 2015

Details

First Name, Surname:	Joris Lückenga
University:	Salzburg University of Applied Sciences
Degree Program:	Information Technology & Systems Management
Title of Thesis:	Reduction of False Positives in Smart Grid Intrusion Detection
Keywords:	Smart Grid Intrusion Detection Anomaly Detection Cybersecurity Pattern Recognition Data Classification Machine Learning Weighted Vote Classification
Academic Supervisor:	FH-Prof. DI Mag. Dr. Dominik Engel

Abstract

The topic of Smart Grid technology and its consequences on security and privacy is a continual discussion in several countries. In order to protect the network infrastructure from security breaches and its possible impacts, several solutions exist and much research is conducted in this area. The following work evaluates and tests a novel classification mechanism for intrusion detection infrastructures, as they are planned for Smart Grid scenarios. The first part covers an overview of Smart Grid topics as well as intrusion detection techniques. For the Smart Grid, common concepts, benefits as well as the security problems are discussed. In intrusion detection, the most popular methods, benefits and problems, as well as existing suggestions for Smart Grid implementations are presented. For further understanding the process of anomaly detection, a basic introduction into pattern recognition tasks as well as a dataset analysis is given. The practical part covers the development of a weighted voting technique and a formula for its weight calculation. Most important steps of the algorithm, a result analysis and constraints of the technique as well as possible parameter improvements will be treated in the last part.

Contents

Affidavit **ii**

Acknowledgement **iii**

Details **iv**

Abstract **iv**

Table of Contents **v**

List of Figures **vii**

List of Tables **viii**

1 Introduction **1**

1.1 The Future of the Power Grid 2

1.1.0.1 Concept and Infrastructure 2

1.1.0.2 Challenges and Benefits 6

2 Smart Grid Security **9**

2.1 Cyber-Threats for Smart Grid Networks 9

2.2 Intrusion Detection in Smart Grid 11

2.2.1 Common Approaches for IDS 11

2.2.2 Current Smart Grid Security Concepts 14

3 Classifier Voting Concept **19**

3.1 Basics for Data Classification 19

3.1.1 Pattern Recognition for anomaly detection 22

3.2 The KDD-NSL Dataset 23

3.2.1 Details and Challenges 25
3.3 Combination of Classification Algorithms 26
3.3.1 Algorithms Selection 27
3.3.1.1 Support Vector Machine 28
3.3.1.2 AdaBoost 30
3.3.1.3 K-Nearest-Neighbor 32
3.3.1.4 Decision Tree 33
3.3.2 Voting Techniques 34

4 Creation of a Voting-Classification system 36
4.1 Implementation and approach 36
4.1.1 Dataset preparation and modifications 38
4.1.2 Testing of Classifiers 43
4.1.3 Combination Technique 46
4.1.4 Experimental setup 49
4.1.5 Results 50
4.1.6 Performance Improvement Function 57
4.1.7 Simple voting techniques 59
4.1.8 False Positive Reduction With Weighted Voting 60
4.1.9 Observed constraints 61
4.1.10 Vote-Classification in a Distributed Environment 63
4.1.11 Creation of a Hierarchical IDS Communication 64

5 Conclusion 67

Bibliography 70

List of Abbreviations 76

List of Figures

1.1 Smart Home Concept [8] 5
1.2 Smart Grid Three Layer Network Architecture 7

3.1 Feature space for Salmon and Sea Bass measures [11, p.7] 20
3.2 Decision boundaries for linear and complex model [11, p.7/8] 21
3.3 Packet types of training and testing dataset 26
3.4 Decision boundary creation with SVM [37] 28
3.5 Kernel-Trick for SVM algorithms [37] 29
3.6 Complex decision boundary of a SVM 30
3.7 Boosting algorithm [30] 31
3.8 AdaBoost decision boundary with 200 estimators 31
3.9 kNN decision boundary with different K values 32
3.10 Boundary example of a decision-tree algorithm 33
3.11 Example of an optimal vote scenario 34

4.1 Graph of the weight balancing function 50
4.2 Chart of detected and existing intrusions with vote classification 52
4.3 Weight function graphs of different parameters 58

List of Tables

3.1 Table of KDD-NSL Features 25

4.1 Example of a confusion matrix 45

4.2 Classification results of each classifier with NSL-KDD training set . . . 46

4.3 Classification results for a 50% split of the NSL-KDD test-set with A-tan scaling 51

4.4 Classification results for a 50% verification-split of the NSL-KDD test-set with A-tan scaling 51

4.5 Classification results for a 50% split of the NSL-KDD test-set with normalized scaling 53

4.6 Classification results for a 50% verification-split of the NSL-KDD test-set with normalized scaling 53

4.7 Classification results for a 50% verification-split of the NSL-KDD test-set without scaling 54

4.8 Classification results for a 5% verification-split of the full NSL-KDD set with A-tan scaling 55

4.9 Classification results for a 5% verification-split of the full NSL-KDD set with normalized scaling 55

4.10 Classification results for a 5% verification-split of the full NSL-KDD set without scaling 56

4.11 Achieved performances for vote-classification 57

1 Introduction

With the growing energy consumption and the commitment of using renewable resources for electricity production, the infrastructure of energy systems started to change. The most visible changes were increasing households with solar panels, investments in wind parks and the construction of different types of power plants using renewable resources. With the looming era of electric mobility and the growing demand for these cars, the power grid has to meet new requirements and challenges in the future: The demand for electricity can further increase, clean energy sources like solar and wind energy are not unconditionally available when needed, and demand peaks can cause energy shortage as well as high costs for producers.

As a solution to achieve an intelligent distribution and management of power resources, the concept of Smart Grid and Smart Metering was developed. This concept is based on the possibility to monitor consumption of end-consumers and control power generators, substations as well as other power grid components over a communication network. The Smart Meter, installed at the consumers site, is able to receive billing rates and manage electronic devices depending on current electricity prices. Another function of a Smart Meter is the possibility to cut the power, in case invoices are not paid or prepaid credits are depleted. These capabilities are very useful and offer a whole new way of power management. Nevertheless, Smart Grid functions may risk the privacy of customers and power grid control components offer a new set of possible attacks for hackers. If an attacker is able to compromise the Smart Grid functionality, he would be able to spy on consumption behavior, gain financial benefits or cause damage to power networks, companies or even countries. Due to this vulnerability, the privacy of consumers and the security of grid components must be assured. For this reason, the following work focuses on reinforcing the security of the Smart Grid, by improving the performance of an existing Intrusion Detection System (IDS) concept.

1.1 The Future of the Power Grid

The Smart Grid is currently a large discussion in politics and economy, because this technology could clear the way for a more ecologic future. As stated in [2], the US invested $4.5 billion for the Smart Grid development. The European Union encouraged member states to evaluate the potential benefits and mandated for a law, which obligates the use of Smart Meters by the year 2022 [14]. This is a very important and supporting condition for the Smart Grid development. By passing laws for the power grid, a fast transition to a new, modernized power grid is possible. Nevertheless, many functions are still in discussion, not yet realized or laws have to be determined. Currently, a lot of research is conducted to find ways for either customers, power suppliers and grid administrators to avoid law infringements, without losing the benefits and functionality of a Smart Grid network. At the same time, privacy as well as security should be provided for each party. The contribution with this work is to further improve detection systems in order to enhance current solutions and technology in the security area. The following sections will give an overview of the Smart Grid concept and its benefits.

1.1.0.1 Concept and Infrastructure

The power grid combines many different types of power plants. Nuclear and coal-power stations, for example, are in the category of base load power plants, which mostly run at maximum capacity with high and constant energy output. They are considerable investments, produce a large amount of electricity and have high maintenance costs. It is also very expensive and impractical to use nuclear or coal plants as backup-power resources, since it takes long to shut down or start electricity production. Every minute of disabled reactors produce high profit losses for the bearer. In addition to that, the

ecology of these power sources is widely contested and efforts are made by several countries to abstain from this type of electricity generation. Nevertheless, the great amount of power output and the increasing demand for cheap electricity led many countries and energy producers to the decision to further invest in nuclear or coal power plants. Alternative base load stations can be fuel-oil, hydroelectric, biomass or natural gas plants. Despite those options, costs are often higher and resource prices or restrictions exacerbate the construction. Depending on the plant construction and power generation method, some of these stations can also be used as load following power plants. These generators are used to follow the growing electricity, based on the time of the day. For example hydroelectric plants are active during the day and evening, when the demand for electricity is considerably higher. When the required power decreases during the night,the water turbines are closed and the water is dammed up. Peaking power plants are used when base and load following stations can not meet the current demand. The costs of kilowatts per hour of those stations is considerably higher, but the supply feed can be established in seconds or minutes, depending on the type of generator. The generators used for peak-demands are often hydroelectric or gas turbines. As it can be derived from those different types of power generation, it is a complicated and challenging task to manage and distribute electricity. In addition to that, a growing number of renewable energy plants and also devices to store electricity, which gain more and more popularity, have to be managed. Further information on the power plants and distribution can be found in [10].
Another important aspect of distributing energy is to measure the consumption and bill a customer properly. This is required to provide a fair business and precise metering is important in order to maintain a stable load on the grid. As high peaks in electricity demand are rather expensive and base-load power is cheaper for the energy provider, the usual prices in several countries vary only two or three times a day. Higher costs for the customer are often billed during the day and early evening, lower costs during night times. To measure the consumed amount of electricity, there is still technology from the first watt-hour meters used. This method of consumption measurement was invented in 1888. Nevertheless, the technology is still wide spread and used in large parts of the world. A pioneer considering electricity metering is South Africa, which

started constructing low cost digital meters in the early 1990s. Due to the rural populations and problems with billing customers, this solution helped the providers by using prepayment meters. Customers could buy credit for their electricity from ATM machines and refill their meter with a code. This avoided problems with customers which did not have a proper bank account or refused to pay their electricity bill. They also discovered that digital meters were more cost efficient and customers started paying more attention to their energy consumption. Brazil, Russia and also the UK followed this example and a transition to the new meters in many other countries is imminent. Ross Anderson conducted much research on this topic and more information can be found in [2].

The idea of Smart Grid is to modernize the whole power infrastructure. Since the power grid grew in many steps to the distribution network of today, many devices from different decades are integrated in the network. Especially in the last years, with large investments and new inventions in the sector of renewable or green energies, the power grid has to meet new requirements. One important problem is, that current energy load in the power grid is distributed based on estimations by the power providers. A factor which adds up to the complexity of this task are wind- and solar-parks as well as private power generators. These plants started contributing to the power network during the last years and the energy production can often vary within seconds. This affects the task of balancing the load in the power network and makes it more and more difficult. To manage all those additional factors, it is very important to control stations remotely and gather enough data on current consumption as well as production. Also, an adjustment of the electricity prices,which is based on the load, would be beneficial for power producers and consumers. Those and many other reasons led to the concept of creating an intelligent power network, which is able to monitor current consumption of every household. Further, home devices which can be equipped with a control & communication microchip, like washing machines, heaters, refrigerators and electricity storage mediums could be triggered remotely by a Smart Meter. A concept of a Smart Home implementation is shown in Figure 1.1.

The core-element of the concept is the Smart Meter device, which receives billing rates from the power provider and controls devices based on the prices. On low rates, the electric car can be charged, the cooler and washing machines can be activated etc. If no energy is needed or the solar panel produces enough electricity, it can be stored in a large storage battery. On high rates, it is also possible to use the vehicle or storage battery to sell electricity to the provider, since high billing rates are often due to power shortage. Otherwise the stored energy could be used to bypass expensive rates.

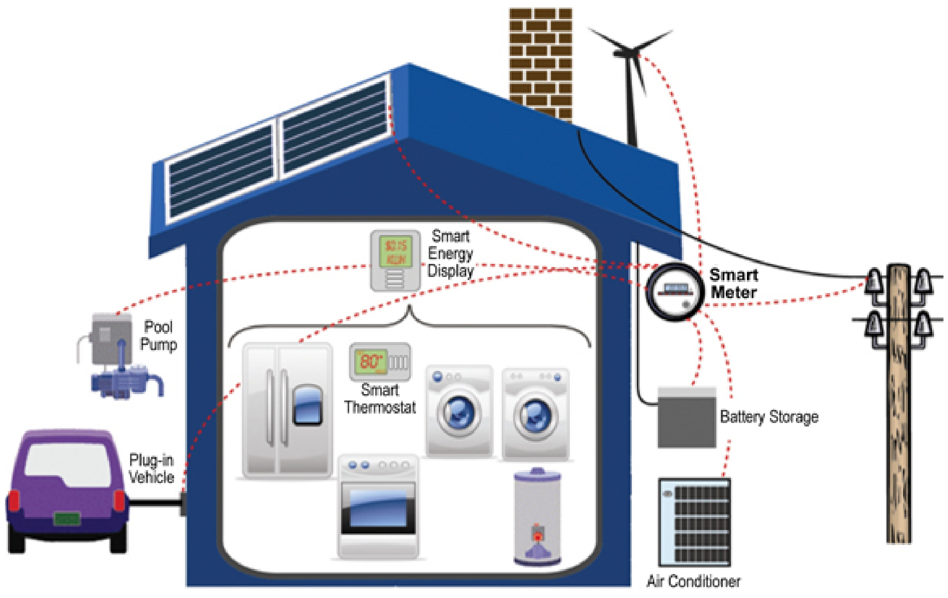

Figure 1.1: Smart Home Concept [8]

Besides the home devices of a customer, also the power grid needs to be equipped with modern communication systems, control modules and monitoring to enable the best possible coordination of energy resources and storage possibilities.

To enable a scenario like described above, new control- and communication equipment, as well as Smart Meters have to be installed and interconnected with the power provider. Not only billing rates, also the current consumption of the costumer needs to be monitored precisely and sent to a control instance, in order to achieve best results. Many different concepts exist on how to establish the communication with the power provider. The work of [34] suggests to implement Smart Meter communication with power & service providers by using mobile Internet connection. Using the private Internet connection of the customer would be possible too, but it is not very convenient, since customers could cut the connection and systems may require a completely new setup, when a client is moving out of a place or changing providers. Apart from those examples, there are many other architecture suggestions of different parties and experts all over the world. A survey on a selection of different concepts can be found in [20]. A very popular approach and also the concept this thesis will consider, is to use a wireless mesh network or grid with distributed communication nodes. A detailed description of the network architecture can be found in Section III A in [48]. This concept is based on a three layer network architecture. The smallest instance is the Home Area Network (HAN), which is basically the architecture presented in Figure 1.1. Every device connected or interacting with a Smart Meter is a member of the HAN and usually only a small amount of people or devices will contribute. The connection to the appliances can be achieved in many different ways, e.g. over WLAN, LAN, Bluetooth etc. The next instance is the Neighborhood Area Network (NAN), which pools HANs and interconnects them. The exit and entry points of this network is either a Smart Meter of a HAN or the NAN switch. Depending on the applied architecture, there are several different ways to establish the connection to the NAN switch. Also it is possible to connect the NAN node with control- and measuring modules of a local power generation plant, e.g. a wind wheel or a small hydroelectric power plant. The largest part of the network is the Wide Area Network (WAN) which interconnects other NANs, power plants, transformer- and control stations as well as the power provider. An illustration of this architecture is shown in Figure 1.2.

On the left, the HAN Network connects storage mediums as well as other electric devices, which are located in or near the house. The exit point is the Smart Meter, which has a connection to the NAN network switch. The NAN network, which is located in the middle, has other HAN networks and a wind wheel connected to the node. The last instance on the right is the WAN network, which interconnects several NAN networks, large power plants and the provider.

1.1.0.2 Challenges and Benefits

With the stated scenario of an intelligent power grid, many benefits can be gained from its implementation. Load balancing is a challenging task and power providers

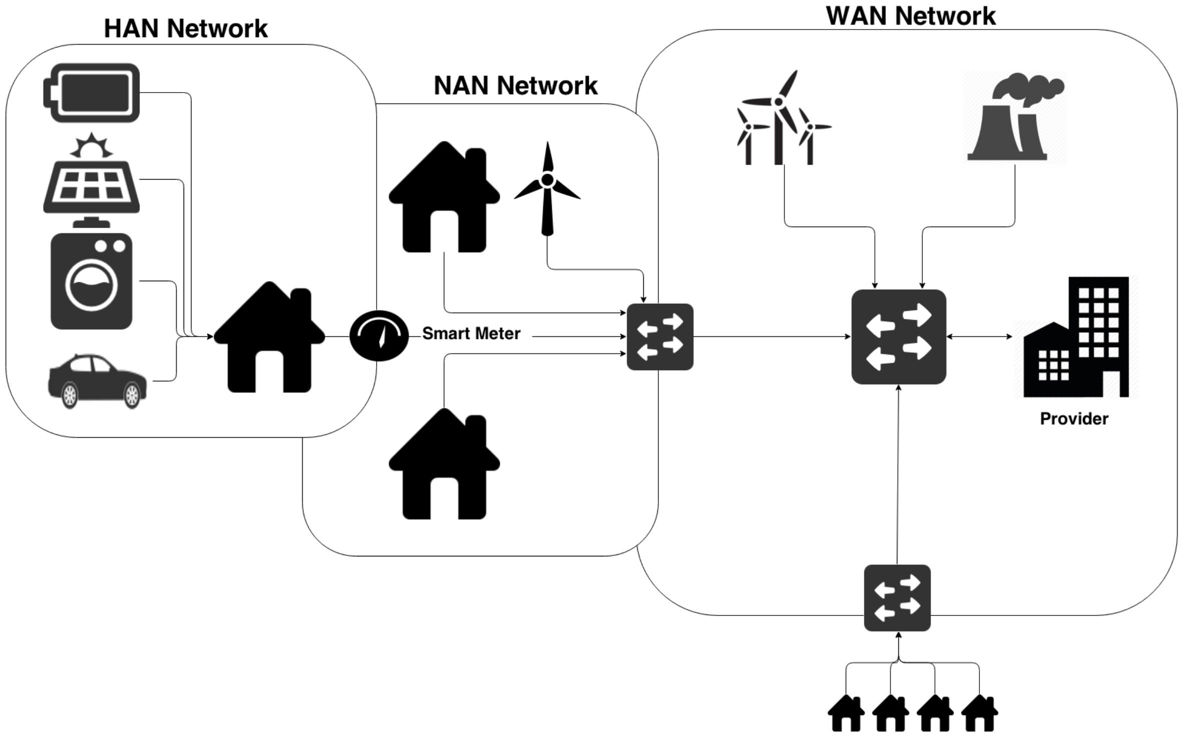

Figure 1.2: Smart Grid Three Layer Network Architecture

must be constantly aware of demand fluctuations in the grid. Therefore, idling backup generators or storage systems are used. This is a countermeasure for voltage drops (Brownouts) or complete power blackouts, in order to enable power providers to react quickly. With Smart Grid technology, the monitoring options and the added flexibility of controlling the grid can help to make the grid more cost effective and reliable. Especially electricity storage systems would highly contribute to the grid stability. Charging car- and storage batteries can be accomplished with low costs when the grid load is low or when generators of renewable resources are running. This way, the electricity provider does not have to further decrease the base load and renewable resources can be used more efficiently. In the other way round, the stored electricity can be used during demand peaks, to avoid high energy costs. Also customers could profit from flexible electricity prices, in case they equip their homes with Smart Meter controllable devices or adapt their consummation behavior to the billing rates. Apart of these few examples, there are many more. Several scenarios can be found in [15, 35, 47].

To realize this power grid, many questions arise with its implementation. The communication infrastructure has to be useful and convenient, it has to be determined which technologies work best together and how interoperability can be assured. Also there has to be decided, how legacy systems can be bound into the system and of course - how the data can be handled in terms of storage, security and privacy. As stated in several resources [13, 24], the data collection, which comes with Smart Grid implementations, is a big concern for privacy and security reasons. For example, personal data collection and also the possibility to remotely cut the power, e.g. in case an electricity bill is not paid, violate European laws of human rights. A set of law violations of a theoretic Smart Grid implementation can be found in [28]. In addition to that, the Smart Grid infrastructure offers a whole new set of attack and espionage scenarios for hackers. More detailed information on the Smart Grid security and possible attacks will be discussed in Chapter 2. Even if there are contrary opinions and surveys on the Smart Grid idea and its implementations, the power grid has to be updated with communication networks in order to meet the requirements of power supply in the future. Also, as it can be found in [21, 46], many projects and implementations exist already. To ensure a safe change for the power grid, research has to be done in order to avoid errors and flaws in the Smart Grid. The past and present has shown in many cases, that the implementation of security systems to prevent attacks in communication networks is inevitable. This work will present an intrusion detection method which could be applied for Smart Grid scenarios and also other different network architectures and applications.

2 Smart Grid Security

In the last years, the motivation of hackers has changed from “having fun” with exploiting software vulnerabilities to more sophisticated and specific attacks on targets to gain financial benefits. Allen Harper describes these changes in hacking behavior in his book [19, C. 1]. Since the Smart Grid requires large communication and control infrastructures, it offers a large variety of intrusion scenarios. Impacts on compromised hardware in the Smart Grid can range from minor monetary losses to threatening situations which can harm lives. The following sections will describe a set of attack vectors and give detailed information about intrusion detection techniques.

2.1 Cyber-Threats for Smart Grid Networks

Since the beginning of electrical power distribution, many issues with metering were discovered. In the early 1990s, overloads, for example caused by lightnings, made the electricity meters run faster and customers started to steal energy by looping a wire over the meter. With more sophisticated digital meters, as they appeared in South Africa around the 1990s, different approaches were found to steal energy. It is possible to recharge these types of meters with a 20-digit code. An encrypted command is then send to the customers meter and the credit is replenished. Even though this system seems to be safe, thefts were made in the distribution chain of the recharge numbers. For example, tampering a vending machine allowed the raider to get free recharge codes. In addition to that, exploitable vulnerabilities were found in the systems software. One example is, that a brownout of several meter types could set the credit to maximum. In order to cause those brownouts, people started throwing chains over electrical 11kV feeders. These examples show, that even with digital technology and securely acting implementations, attackers can and will try to manipulate the Smart Grid infrastructure in order gain financial benefits. One specific Smart Grid scenario, in which an attacker tries to lower the billing rates is shown in [29]. This scenario describes the procedure of packet-manipulation to inject false data into the Smart Grid network. As a result, billing rates are compromised and other clients have to pay more. Another problem is the intention to weaken a company or country by invading communication networks to limit or stop critical services. Current events showed that hacking activity is used for espionage and can be used as an instrument to threaten or damage countries. For example, as shown in [25], the North Korean government trains and recruits skilled hackers, possibly to gain more potential in cyber-warfare abilities. Common viruses or attacks, like Stuxnet or countless examples of exploits available in the Internet are menacing threats for the Smart Grid security. Especially a virus which is active in control mechanisms and SCADA-Systems can cause serious damage. SCADA-Systems are data-acquisition and control systems which are responsible for regulating and supervising a large amount of power grid devices. In a survey of the Anti-virus company Symantec, attacks on the energy sector were gathered and presented. It becomes quite clear, that the power grid is a very popular target: "In the second half of 2012, the energy sector was the second most targeted with 16 percent of all the targeted attacks [...] The attackers tend to go after valuable information [...] but the sector is also a major target for sabotage attacks"[6, p. 19]. Many other different types of Smart Grid attacks can be found in [45, 17, 29]. With the possibility of controlling electrical devices and cutting power supply, the Smart Grid functionality must be ensured and secured. Great threats, which even can harm lives, for example when electronic medical equipment is in use, become reality if attackers are able to compromise control modules of the power grid. Further information on the importance of Smart Grid security, challenges and requirements are presented in [24, 49, 2, 3, 13]. The basis of all these attack-types are manipulated packets, which contain a malicious

payload or cause system faults. Therefore it is essential to filter the traffic and control the packet flow.
To defend against the above mentioned actions, intrusion detection concepts to ensure the Smart Grid security are developed. An overview of IDS techniques and approaches can be found in [44]. As IDS are also a well known topic in Internet technology, concepts from these systems are adapted to Smart Grid networks.

2.2 Intrusion Detection in Smart Grid

There are many different approaches on how to protect systems against attacks. Most common are firewalls, which block ports of incoming and outgoing traffic. They are either installed on hosts or they are used to secure incoming web-traffic from the internal network. This is more an approach to block vulnerable spots and dangerous protocols to prevent a successful attack. Another way to uncover attacks is to attract hackers by using a vulnerable system, called a honey-pot, which has security flaws on purpose, but does not carry any sensitive information. If an attack is conducted, the attacker can be uncovered over the logging or the misbehavior of the system. Security authorities become aware of an attack and will conduct countermeasures to stop further attempts. All of these techniques are not able to detect a specific attack when the packet is sent to a target. Therefore, intrusion detection systems are needed, which either work on a host,on a network or are integrated in a firewall. These systems inspect packets and sessions of a communication and use different methods to recognize attacks. The following sections will describe how intrusion detection systems work and which techniques are used. Furthermore, current solutions and concepts for the Smart Grid are presented.

2.2.1 Common Approaches for IDS

The general idea of intrusion detection is to identify an attack, either before a malicious packet reaches its destination or shortly after it. A desirable goal of the concept is the identification of an attack to initiate countermeasures as soon as possible. If that is possible, the packets can be discarded and a potential hacker can be isolated from the network. The following approaches are most commonly used:

Misuse Detection is based on the concept of comparing a packet against a malicious behavior. The system is trained for attack-patterns, e.g. when a protocols tries to execute sensitive system commands. These patterns are often abstracted from known attack techniques. Misuse-detection systems contain a signature-database with common malicious patterns. If a packet arrives, it is scanned and compared against the database. In case a packet matches an entry of the database, the communication is discarded and an administrator is informed.
The downside of this is the limitation to known attacks. If a hacker uses a so-called zero-day attack, the malicious packet is not known yet and probably uses a new security flaw to intrude the system. A misuse detection system is most likely not able to identify the attack and allows the packet to pass without any issue. Especially in Smart Grid implementations, where attack patterns do not exist yet or are not available in a large variety, this can be a crucial security issue. As shown with the example of digital Smart Meters in South Africa, new technologies often have many unknown flaws and hackers may be able to find security gaps more easily. Therefore, a misuse detection system is less likely to be an effective IDS in Smart Grid applications during the first years of usage. In case this type of IDS is deployed in a power grid network, it would be best to combine it with other techniques, in order to complement it with a detection method for unknown attack vectors.

Specification-based Detection is a derivation of a misuse IDS but uses a different approach to find rules for the communication behavior. The specified actions in a protocol or systems are usually defined very precisely. Based on the constraints and the behavior that is described in the specification, a rule-set can be extracted. This means that packet parameters or actions of system components can only act in the defined range. Especially when using simple systems and only few protocol types, this method can be very effective to protect against intrusions. The more complex a system or protocol is, and the more variety of communication methods are in the network, the more challenging is the task of designing a specification based IDS. In addition to that, there is no guarantee that the specification does not leave any room for security flaws. As an example, the Heartbleed vulnerability used legitimate Heartbeat packets to retrieve data from the server RAM and therefore could be used to steal private keys and user session cookies. The problem was, that just one variable in the packet was not checked correctly, but the impact on Internet security was enormous. Even though this issue was implementation based, those flaws may also occur in a specification. More details on the Heartbleed vulnerability can be found in [12]. Therefore a specification based IDS has to be designed very carefully and this task can be more and more challenging with increasing complexity. Nevertheless, this approach can be very effective for the Smart Grid, in case implementations are simple or control mechanisms can be generalized easily. This is often the case when security is integrated in the design process in the early beginning. Due to the problem that many companies have already created specifications and are also running Smart Grid devices in several projects, the implementation of a specification based IDS may be too difficult afterwards. In addition to that, every company and its own implementation will require a specific IDS and this would cause problems with compatibility.

Anomaly Detection is used to train a system for normal traffic. To achieve this, several methods can be applied and will further be described in the following chapters. Generally, the system will “learn” the normal communication behavior of the network. Depending on which approach is used, the system can also be trained with data of anomalous behavior to enhance detection chances. That way, the IDS can decide, based on a set of parameters, if a packet has normal behavior or not. This is a great advantage compared to the misbehavior system, since zero-day attacks might not be able to fall through the detection mechanism. If suspicious parameters occur in the packet payload, the IDS can detect an anomaly. The detection approach can be used in any network, completely independent from existing implementations and protocol standards. This is a critical aspect, because Smart Grid implementations already exist in many countries and anomaly solutions simplify the deployment of IDS in an existing network structure.

The problem with anomaly detection is the vague border between good and bad behavior of packets in a network. It is a complicated task to design a line between the classification decisions. The result of this is the very certain appearance of false classifications. A "bad" packet might be able to slip through the detection mechanism, and a "good" packet might be classified as an intrusion. Those errors are called false positives (FP), in case a normal packet is classified as an attack, and false negatives (FN), in case an attack is not detected. They can occur, for example when the IDS has never learned the type of packet parameters or payload before. High rates of FP or FN cause inefficient defense mechanisms, detection overflows and system misbehavior. The problem of occurring miss-classifications is accompanied by high sensitivity to FP or FN in the Smart Grid network, due to the large amount of machine-to-machine traffic with important control flows. As a consequence, an efficient anomaly detection mechanism with very low FP/FN rates is required, in order to assure flawless Smart Grid communication.

There are several derivations and other methods for intrusion detection available but usually the basic principle of anomaly- or misuse detection is applied. A set of current concepts on Smart Grid IDS will be presented in Section 2.2.2. More and detailed information on this topic as well as other approaches can be found in [44] and [1, C.2].

2.2.2 Current Smart Grid Security Concepts

Several concepts and suggestions to secure Smart Grid communications have already been developed. The work of A. Murillo gives a review of some anomaly detection methods in Smart Grid systems [4]. This section will also present a set of different popular concepts and alternative ideas of securing a Smart Grid network. Due to the large variety of suggestions and methods which can be applied in the Smart Grid, the following will only give a short summary and a small assortment of ideas to give an overview of current research. Also restrictions and constrains for each work will be presented. Some concepts might be adapted and integrated into a Smart Grid security concept in order to achieve the best possible solution.

Mobile WAN Connection Concept

In the work of [34], the Smart Meter is basically the core-element of all functionality. Due to the mobile Internet connection, which is used to communicate with the power provider, a network infrastructure besides the HAN is not required. Also, the Smart Meter is considered a multi-functional device which contains a firewall and provides Power Line Connection (PLC) Zig-Bee or WLAN for HAN communication. To ensure the security and integrity of the connection, a tamper-resistant cryptoprocessor is used. The device will also have cryptographic algorithms and functions to support a public-private key infrastructure. The concept also suggests that, apart from the sender and receiver part of the transmitted data, packets should be fully encrypted. Using encryption would make this system safe, if the methods are applied and implemented correctly. It also would make IDS rather difficult, because the package content could not be read and checked against attacks. Another issue which Anderson states in his work [3], is the problem of key compromising. If the private key of a provider is acquired, an attacker could remote control a large set of meters and would be able to cause large electricity blackouts. Nevertheless, the concept provides good security measures and does not require the construction of a NAN and WAN infrastructure.

Model Based IDS

An approach of F. Tabrizi and K. Pattabiraman [42] focuses on Smart Meter security. It suggests a concept which enables the Smart Meter software to detect unauthorized function calls. This means that an intrusion is not detected by an independent device or filter. Instead, the Smart Meter software checks itself when code is executed. Therefore, the Smart Meter software and its architecture has to be specified and additional code annotations have to be added to the source code in order to enable the functionality. These annotations specify important security functions and in case a sensitive function is called, there will be a check against an abstract rule-model, if the call is legitimate. Tests only could be conducted by trying to insert bad function calls and check if the IDS worked. Four scenarios were tested successfully and the system seems to work efficiently. Another advantage is, that if only sensitive functions are monitored, the overhead for the CPU, which is added due to the integrated detection mechanism, is not very high. The downside of this concept is that this solution has to be engineered into the system before it is sold or firmware updates have to be enabled. Also it does not protect other assets on the grid. Despite this, the concept would be a great additional “last barrier” for attacks against Smart Meters, especially when encryption is used. The encryption would not hinder the detection mechanism because the package is decrypted before. Afterwards, resulting actions are taken with the aid of function calls, which are checked by the IDS. The method could also be adapted to secure other hardware assets in the grid.

Rule Based IDS

In the work of [32], a Behavior-Rule Based Intrusion Detection System is suggested. The argumentation here is that anomaly detection methods can not avoid false classifications and still have too many false positives in the current research. Therefore it is argued, that they are not suitable for Smart Grid implementations. Instead, the work suggests a derivation of a specification based technique. In order to specify constraints for the possible actions of network nodes, a table of behavior rules is created. A rule can look like the following: “if a certain demand is above the threshold, power generation is increased”. This would be valid behavior. Any derivations from this rule would be considered as an attack. Every node has a monitor and a trustee to be able to control the behavior of each other. This approach can be very effective, since known as well as unknown attacks would trigger invalid behavior, like lowering the price, even if the demand is high. The downside of this is similar to the specification based concepts. If complex implementations already exist or if there is not enough effort done to be able to implement this rule based concept, the creation of this type of IDS might become too complex.

Applying Domain Knowledge for IDS

One method of enhancing anomaly based intrusion detection is presented in the work of [27]. This concept suggests using Fuzzy Logic systems in order to represent human domain knowledge. To achieve this, several rules have to be defined. One example could be: “if the number of protocols is high, then an attack is more likely”. This can be the case if a system only uses a small variety of protocols and it is rather unlikely that a large protocol variety occurs. This knowledge can be mapped to a fuzzy rule, which controls the sensitivity of an anomaly detection algorithm. In case the detection sensitivity is high, packets are more likely to be considered as an attack and this will trigger more findings. It will also increase the likelihood of false positives, but since an attack might occur, it would be useful to find every malicious packet. The other way round would happen when the number of protocols is low. In this case, less detections are triggered. This concept could be used very efficiently, when good human domain knowledge rules can be found for a system. It also could be combined with an already existing implementation, in order to enhance detection accuracy.

Smart Grid Distributed Intrusion Detection System

One promising and popular concept for detecting anomalous traffic in the Smart Grid is the Smart Grid Distributed Intrusion Detection System (SGDIDS) developed by Yichi Zhang, Lingfeng Wang, Weiqing Sun, Robert C. Green II, and Mansoor Alam [48]. This concept uses interconnected, distributed IDS nodes in a network infrastructure as it is illustrated in Figure 1.2. The concept was tested with a modified KDD-NSL dataset and three different anomaly detection methods were used. Also, clustering algorithms CLONALG and AIRS2Parallel were used in order test unsupervised classification efficiency. The basic principle is to have IDS systems in every network layer, which are able to communicate between the IDS nodes in the hierarchy. In case a node in the lower level of the hierarchy can not classify the data, the packet is passed to the next instance, which has more powerful detection algorithms. Since this approach uses a popular infrastructure as well as a widely usable and retrofittable anomaly detection, this thesis will adapt the idea and further tries to test a new approach on anomaly detection. Due to the sensitivity of the Smart Grid communication, the attention is directed to decreasing the false positive rates.

3 Classifier Voting Concept

This chapter will explain the idea of a classifier voting concept and give an overview on anomaly detection. In order to be able to further describe the concept, a basic overview on pattern recognition techniques and selected algorithms will be given. Also, the important factors which influence the precision of the classification output will be presented. Pattern recognition is a very large, knowledge intensive topic and due to the restricted scope of this work, only a small and basic part will be covered. Further and detailed information on machine learning and pattern recognition can be found in [11].

3.1 Basics for Data Classification

The common technique which is used in an anomaly detection system in order to detect and analyze an Internet packet, is called pattern recognition. It is a branch of machine learning and focuses on detecting regularities in data. To describe how the process works and which problems are involved, an example of the book "Pattern Classification" is used [11, Ch. 1.2]. In this case, a fish plant wants to automate the process of sorting incoming fish into the according species. The two types of fish, sea bass and salmon, should be recognized by optical sensing. In order to distinguish the breed, it is possible to extract certain features from the pictures, like length, width, number and shape of fins as well as the mouth position. It has to be considered that the values of the features vary picture by picture, because of different sizes of the fish, lightning conditions etc. To reduce this occurring noise, preprocessing is used. In this procedure, the data is simplified, but relevant information for the breed prediction should not be lost. Afterwards, the features with the biggest variation between the breeds are used in order to built a model. In this case, the width of the fish and the lightness is selected, because the sea bass usually has more lightness than the salmon. If the features are populated into a two-dimensional space as shown in Figure 3.1, the black dots represent features from salmons and the pink dots represents features from sea basses. The data here is represented as the training data for a classification

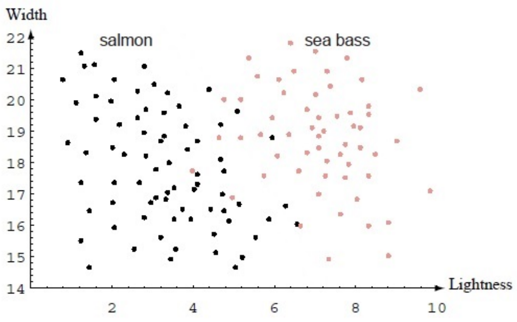

Figure 3.1: Feature space for Salmon and Sea Bass measures [11, p.7]

algorithm. The next step is to build the model, which defines the relationship between features and patterns. This means that a set of features, like a two-dimensional vector, containing the value for width and lightness, is mapped to the pattern salmon or sea bass. Depending on which algorithm is used, different decision boundaries will be created. These boundaries decide, how a new vector will be classified. In a two-dimensional model, those lines can be in different shapes, like linear, cornered, round etc. Figure 3.1 shows how the decision boundaries look like in a two dimensional feature space for a linear model and a complex model. If three dimension are used, the line becomes a plane and if more dimensions are added, the decision boundary will be in the form of a hyperplane. In the classification process, a fish is photographed and the values for its width and lightness are preprocessed and transformed into a vector. The values are populated in the feature space and depending on which side of the decision boundary the dot appears, the classification is made.

After a model is build with the training data, it is usually tested with a new set of data, the testing set. It contains data which has not occurred before in the training

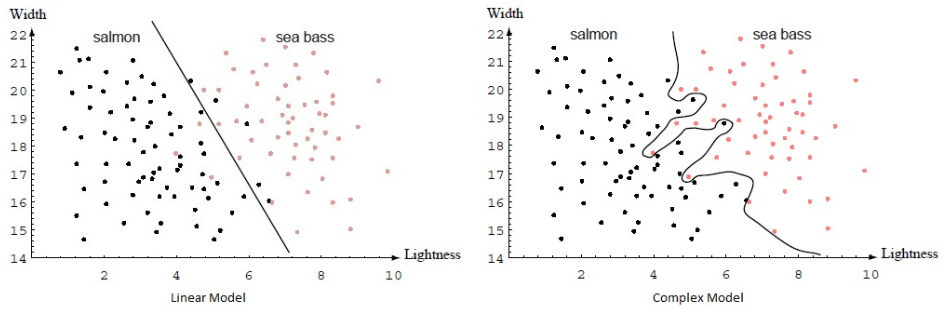

Figure 3.2: Decision boundaries for linear and complex model [11, p.7/8]

set and usually consists of similar data instances. The testing set is used to determine the classification efficiency, so it can be resolved if the constructed decision boundary can also be exposed to new data and still make mostly correct predictions. To create a test set, two approaches exist. The first method is to use a small amount of the whole dataset, for example a 10% or 20% split for testing and the other, larger part of the data is used for training. Another method is to use a test-set which is already provided with the data. This is often the case when popular datasets are used, to enable benchmarking of the algorithm efficiency. When it comes to evaluation, there are many ways to measure the performance and the method also strongly depends on the used data. Most common values are FN and FP rates for two class performance measures. Another way of showing the precision of the predicted output is to populate the data in confusion plots, which show correct and falsely classified data in a matrix format.

As it can already be observed on both the linear and complex model in Figure 3.2, there might be problems with the boundary. In the complex model, no data is classified incorrectly in the training set. But when there is new data populated in the feature space, it is very likely that false classifications may occur. The problem here is called overfitting, since the boundary is matched too precisely to the training set. Another issue is the feature space, which is not able to create a large gap between the classes. This causes some vectors to blur into the other class. In the linear model, there are already cases in which the salmon will be classified as sea bass and vice versa, but the accuracy of this model might be more efficient than the one of the complex model. To determine the efficiency and to check if the complex model might be overfitted, the testing set is used. This example shows how important models, decision boundaries and feature selection are for classification efficiency. Sometimes it also makes sense to use a verification set, which is a new set of data which hasn't been in the training or testing set yet. This is used in order to check if the classification model has not been fitted too precisely to the testing set.

3.1.1 Pattern Recognition for anomaly detection

The example stated Section in 3.1 is simple but the main principles do not differ in most of the anomaly detection approaches. In order to enable pattern recognition, features have to be extracted, preprocessed and a model has to be built. In the case of anomaly detection, the main goal is to train the system with the normal traffic. In order to do this, it is important to select a set of features which will create a cluster for normal data packets and place anomalous packets somewhere outside of the cluster. The outlier-packets can then be detected by a classification system and therefore recognized as attacks. It is strongly dependent on the features and the prescaling method applied, how far away the anomalous packet can be placed from the "normal-cluster". This means the most important task comes down to the feature extraction and therefore finding, which properties differ most between intrusion packets and normal traffic. Since this can be very complicated, a common approach is to gather every feature that can be thought of, and then reducing the data to the relevant features with most variation between the classes. Many different values can be extracted from digital communication, like protocol types, services, ports etc. But since this engineering task is not only implementation specific but also very difficult, this work will use a common dataset to evaluate the classification performance. Using a popular performance measuring method also enables the possibility to evaluate and compare the performance of a classification system. Further information on this will be presented in Section 3.2. After the features are selected, there are many different ways on how to detect outlier-packets. Common approaches and research from the last six years in this area can be found in [5]. Popular techniques for pattern recognition are the use of Support Vector Machines (SVM), Cluster analysis and Ensemble techniques. The methods also vary on which layer the anomaly detection is used and which data is gathered. Either the system is integrated in a machine and the IDS controls the behavior by monitoring device parameters, or the detection system works on a meta level. For example when a set of devices is used and the IDS monitors the communication traffic. Also device status and behavior updates of the machines can be used in order to create an anomaly IDS. Since deploying an IDS as a network node is the most flexible solution and also offers the possibility to upgrade a network infrastructure after its implementation, this approach is used in this thesis.

3.2 The KDD-NSL Dataset

To evaluate the classification performance, the KDD-NSL dataset is used. The files can be downloaded on the website of the Information Security Center of Excellence [36]. This dataset is a modified version of the popular dataset KDD Cup 1999, which can be found on the SIGKDD website [41]. The reason for the name and its existence is, that the data was used in the Third International Knowledge Discovery and Data-mining Tools Competition which was held in 1999. The purpose of the set was to evaluate an anomaly based intrusion detection system, which was developed by the competitors. The data consists of normal traffic and attacks, which can be divided in four categories:

1. Denial of Service (DoS) attacks like syn-flood, to block or overload a targeted system.
2. Root-to-Local (R2L) attacks to gain access to a remote machine. Methods for password or login acquisition are used, for example guessing passwords
3. User-to-Root (U2R) attacks to gain administrator privileges. In this case, a very common approach is using exploits with buffer overflows
4. Probing, to gather system information. This can be done with port scanning and OS-fingerprint techniques

The set was prepared and managed by MIT Lincoln Labs. To gather the data, standard communication was audited and a wide variety of intrusions, which were simulated in a military network environment, was added. For the classification purpose, 41 features are contained in the dataset. A brief description of every feature can be found in the task section of the SIGKDD site [41] and in Section 3.2.1. The set consists of simple features like a packet flag and protocol as well as meta-level, connection-,time-, or host-based features. For example the host-based feature “Same Host” works on a higher level, since it examines the last requests to the same source in a window of 100 connections. In this case not only knowledge of the packet itself but also of past connections is included. This is effective in order to discover several attack types because it also enables to evaluate communication specific features. If only packet-specific attributes are used, additional information about the past connections can not be gained and this might worsen the discovery chances. Even though the features seem to be chosen in an efficient way, some problems exist in the original KDD CUP 1999 dataset. One problem is the large size of over 700 Megabytes, which requires algorithms and systems with high performance. Another issue is the amount of redundant records, which biases the classifier performance. A detailed analysis of these problems can be found in [43]. In order to approach the issues, the NSL-KDD dataset was developed and the amount of records was significantly reduced. An overall reduction rate of about 75 % was possible after eliminating redundant records. With that reduction, the dataset comes down to a size of about 20 Megabytes, which makes it very applicable for experimenting with classification systems. Even though there is still critique about the addressed problems of the CUP 1999 set, which were not resolved with the KDD-NSL development, the NSL-KDD is widely used by several researchers. Details for of the unresolved issues are stated in [23]. This popularity is due to the reduced size and the enhanced applicability, which is offered by the dataset. It has to be considered that using data in this way is only a first step for general performance evaluation. It enables to compare results and helps to determine the efficiency of the proposed method, which also applies for the implementation tests in this thesis. Even though the solution is suggested for a Smart Grid implementation, regular Internet traffic is used for the evaluation. This is due to the lack of available smart-grid communication data. A dataset with attacks and Smart Grid traffic is not yet available.

3.2.1 Details and Challenges

In order to understand and interpret the results, a detailed analysis of the dataset is necessary. The NSL-KDD consists of 42 features, including the class attribute. Every feature is either numeric, binary or a string value (nominal). Also, the values are either extracted from the packet or contain a "meta-value" of the communication. Table 3.1 shows the available features, their format and type. As it can be observed, there are five same host and three same service features. In addition to that, there are nine features with discrete values. Four are string- and five are binary values. The class-feature is also a discrete string value, but it is not considered as a real feature, since it serves for classification evaluation.

Features with type and format		
duration	[D,Bin] su_attempted	[SHo] same_srv_rate
[D,Str] protocol_type	num_root	[SHo] diff_srv_rate
[D,Str] service	num_file_creation	[SSe] srv_diff_host_rate
[D,Str] flag	num_shells	dst_host_count
src_byte	num_access_file	dst_host_srv_count
dst_byte	num_outbound_cmds	dst_host_same_srv_rate
[D,Bin] land	[D,Bin] is_host_login	dst_host_diff_srv_rate
wrong_fragment	[D,Bin] is_gust_login	dst_host_same_src_port_rate
urgent	count	dst_host_srv_diff_host_rate
hot	[SHo] srv_count	dst_host_serror_rate
num_failed_login	[SHo] serror_rate	dst_host_srv_serro_rate
[D,Bin] logged_in	[SSe] srv_serror_rate	dst_host_rerror_rate
num_compromised	[SHo] rerror_rate	dst_host_srv_rerror_rate
[D,Bin] root_shell	[SSe] srv_rerror_rate	[D,Str] class

D = Discrete feature, Str = string value, Bin = binary value
SHo = same-host feature, SSe = same-service feature		

Table 3.1: Table of KDD-NSL Features

There are different types of KDD-NSL datasets available, which include a training and a test set. The training set consists of 125973 instances, the testing set has 22544. There is also a subset of the testing data available, which does not include data of a certain difficulty level. In this work, the full testing set will be used. In order to analyze the dataset in more detail, Figure 3.3 shows the number of packet types of the two datasets, split into the specific attack-types and normal traffic.

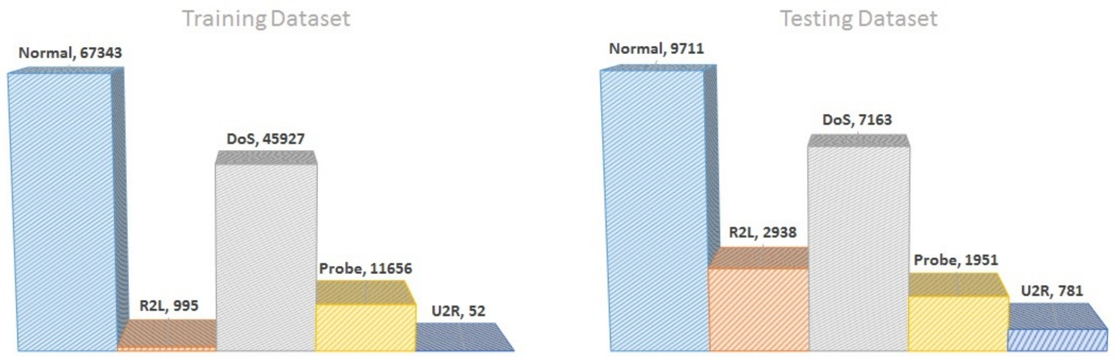

Figure 3.3: Packet types of training and testing dataset

As it can be observed, the Training dataset has many normal, DoS and Probing instances. The amount of R2L and U2R types is rather small, only 52 Instances for U2R and 995 for R2L. In contrast to that, the amount of R2L and U2R attacks in the testing set is fairly large. The R2L instances are three times the size of the training data and the amount of U2R attacks is even 15 times higher than in the test set. This means that an efficient training for those attacks is rather difficult with the provided data. Another complication for the attack detection task is the large variety of trained and untrained intrusions, which are present in the test set. To produce a more challenging set, the developers interspersed attack methods which are not contained in the training data. This requires the IDS to detect unknown intrusion patterns, in order to be efficient. The intention of this is to test the algorithm for its ability to detect zero-day attacks. Therefore, it is very challenging for an algorithm to produce a good performance, but the more challenging the dataset, the better an evaluation and comparison is possible. Since unknown attacks can be the most harmful ones, this encourages the production of algorithms, which may be able to detect unknown intrusions in the future.

3.3 Combination of Classification Algorithms

The following sections describe different algorithms which are implemented in a classifier voting scenario for an IDS. To be able to classify data in a voting scenario most efficiently, different approaches to classify the data are used. A set of algorithms produces a variety of different classification outputs, which then can be combined with voting. The classifiers are chosen based on their performance and learning strategy.

By using the most efficient algorithms, a voting mechanism may achieve best results. Combining different learning strategies can also play an important role for efficiency and the impact on the performance is examined. It has to be considered that a supervised classifier may perform best in a testing scenario with a defined database, because different attack strategies and types are already known and classified correctly. The supervised classifier can be trained with a clearly defined good and bad behavior. Similar attack patterns as trained in the KDD-NSL dataset may be detected, even if a zero-day attack is present in the traffic. When an IDS system is deployed in a Smart Grid scenario, a predefined and classified attack-database is not yet available. Despite this problem, an unsupervised classifier does not require training for attacks and will be able to decide without indication, if a packet is considered as an intrusion or not, for example by clustering. Therefore an unsupervised algorithm might be useful in the current situation, because it only needs normal behavior data to be trained. Nevertheless, companies and security specialists may be able to create a Smart Grid dataset for supervised learning in the future, which would enable to use the generally better performing supervised classifiers. This thesis will also focus on supervised training algorithms, since benchmarks are more easy to create and new concepts mostly adapted this type of training.
To reach the desired goal of reducing a FP-rate of an algorithm, the focus is on improving the performance of individual classifiers with a combination of learners. It is not the goal to achieve better results than research has attained before with the KDD-NSL dataset. It also has to be mentioned, that the feature extraction is the most crucial factor for the effectiveness of classifiers. If it is possible to divide bad and good behavior in Smart Grid traffic by using a set of features containing high variance between normal and anomalous traffic, packets can be easily distinguished. This could enable the possibility to detect intrusions with a great probability and even unsupervised clustering algorithms may then be used as an efficient mechanism to detect intrusions.

3.3.1 Algorithms Selection

In order to understand the different classifiers and the learning techniques, the following section will present a set of selected classifiers, which contribute in a voting scenario.

The explanation of the algorithms will only cover the basics and will not be illustrated with the NSL-KDD data. The dataset used in this work contains 42 features and since every feature is used, the dimension in which the classification is carried out is the same as the amount of features. However, this is not presentable in order to explain the algorithms,and therefore, examples with two-dimensional data will be used. The methods presented were implemented and tested, and can also be applied to more than two dimensions. In the case of multidimensional data, the decision boundaries will form a hyperplane or multidimensional areas instead of simple lines. In the examples shown, the boundaries are always represented as lines in a two dimensional feature space.

3.3.1.1 Support Vector Machine

Support Vector Machines (SVM) are used for pattern classification and regression. This algorithm only uses mathematical procedures to form a decision boundary. The main concept with SVMs is to create a hyperplane, which is able to separate the different classes. The goal of this method is to maximize the distance between the vectors of differing classes, which are located closest to the decision boundary. Figure 3.4 shows the separating planes and the optimal hyperplane in a simplified manner. On the left side, possible decision boundaries are shown, but the distance between the class data-points are not maximized. The optimal hyperplane on the right side is found by the SVM, due to its algorithm.

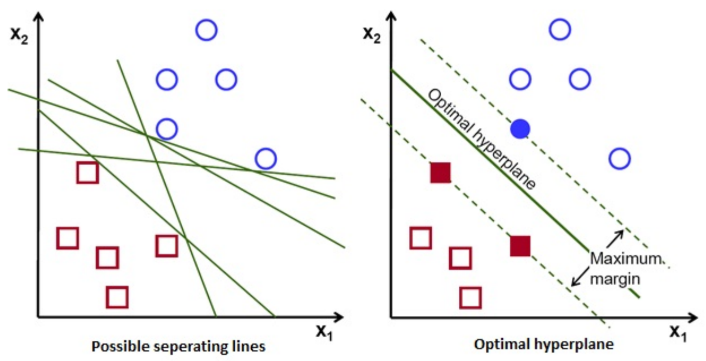

Figure 3.4: Decision boundary creation with SVM [37]

In this case, a linear separation and a two-class classification is used. Since this is only applicable in a simple feature space, which can be separated by a line, more complex solutions for creating a decision boundary are required. Therefore, SVMs can use different kernels to do a so called “Kernel-Trick”. With this method, additional dimensions are added to the feature space in order to make the data linearly separable. This is possible due to the fact that every feature space can be linearly separated, if only enough dimensions are added. When this line in the higher dimensional space is transformed back into the original space, a non-linear boundary is created. It is determined by the kernel selection, in which way and which general shape the decision boundary is formed. It may require just one or even nearly unlimited additional dimensions to achieve this, depending on the feature space used. An example of a Radial Basis Function (RBF)-Kernel and a Polynomial Kernel is shown in Figure 3.5.

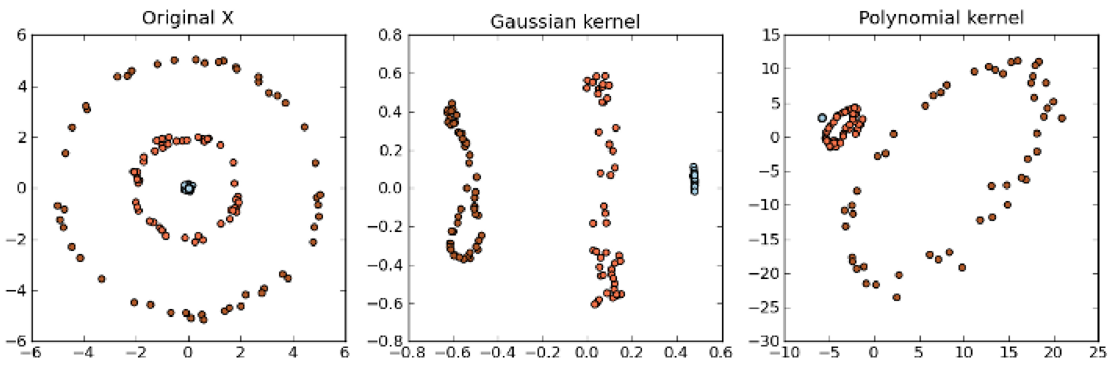

Figure 3.5: Kernel-Trick for SVM algorithms [37]

In this example, the feature space illustration on the left, which consists of red and blue dots in a round shape, is transformed with the Kernel-Trick into a linear separable area. The plot in the middle shows a Gaussian kernel transformation, on the right is the polynomial transformation. It can be observed that in this case, the Gaussian Kernel might be a more efficient method to classify the data, since the blue dots are separated clearly from the red ones. Also, different parameters can be used for the decision boundary tolerance, based on which kernel is applied. This is practical in order to influence the classification performance. These operations can be very complex and finding best fitting values is an experimentation task. The consequence of this technique is also, that the higher the dimensions and instances of the training data, the more calculation intensive is the SVM algorithm. Figure 3.6 shows an example of a complex decision boundary of a SVM.

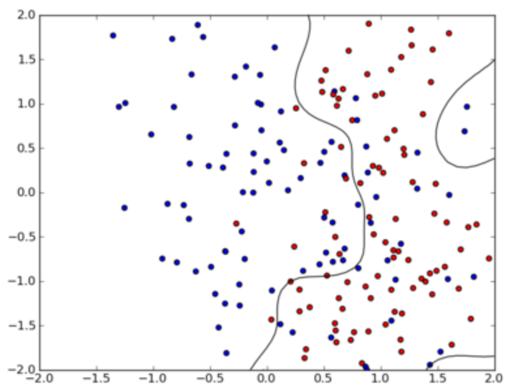

Figure 3.6: Complex decision boundary of a SVM

In addition to that, many different implementations of SVMs exist, which can even handle multi-class problems. Due to the kernel-tricks, training time might be very long. However, the prediction of data-points is often much faster, once the decision boundary is set. More detailed information on this topic can be found in [40].

3.3.1.2 AdaBoost

AdaBoost is short for “Adaptive Boosting” and is an ensemble meta algorithm. It is currently very successfully used as a face-detection method, but can also be applied for other classification problems. It is called an ensemble-algorithm since it uses a set of simple learners which are then combined in a voting method. This method originates from boosting. When learning, a simple classifier, for example an algorithm that creates a linear boundary, like a horizontal or vertical line, is trained with the dataset. If the feature space can not be separated by this single line, there will be miss-classifications in the dataset. The data-points which weren’t detected correctly will then be weighted as “more important”. In the next step, a new model will then try to classify every weighted data-point correctly, which means that every miss-classification on the last step should now be covered with the next model. The data-points which haven’t been detected correctly by the second model, will again be weighted and a new model is trained. That procedure will be done several times and in the end, weights are applied to every model. With combining each simple model with a voting-weight, a decision boundary is created. This procedure is illustrated in Figure 3.2. M1 shows the first model, which has three false classifications. The enlarged “+” data-points show the false classifications on the first model, which shall be covered by M2. M3 shows the third decision boundary, which tries to cover the enlarged false classifications of M2. The circled plus data-points are already covered by M1 and not considered. In the final result, all three models are combined to one decision boundary with weights, as shown in Figure 3.7.

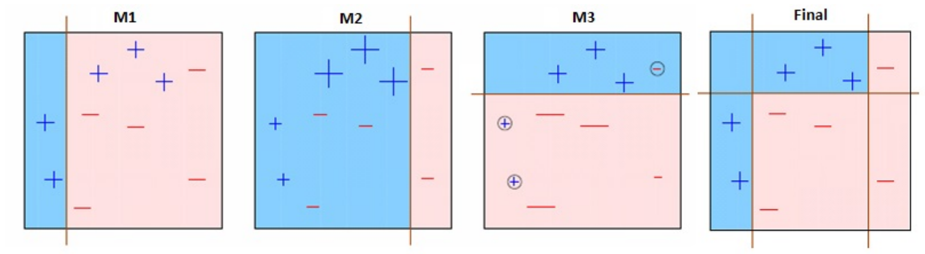

Figure 3.7: Boosting algorithm [30]

The decision boundary form can vary with the chosen models, but most commonly a linear classifier is used, which creates an angular surface. With AdaBoost, the weights are updated in each operation with a combination of logarithmic and exponential functions. The more models, or so called estimators, are trained, the more precise the model can become. As a result, AdaBoost can be a very efficient algorithm, which is used in more and more applications, due to its increasing popularity over the years. A more complex example of an AdaBoost decision boundary is shown in Figure 3.8.

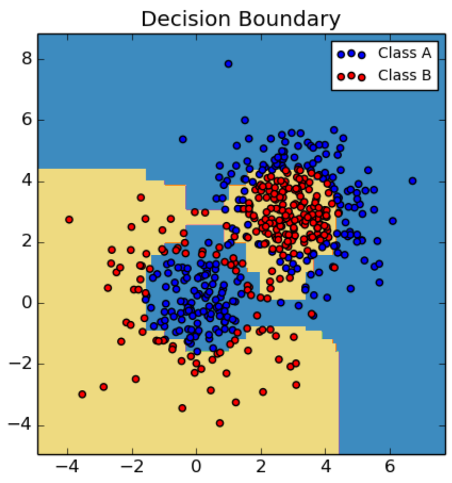

Figure 3.8: AdaBoost decision boundary with 200 estimators

As it can be observed, the decision boundary strongly differs from those of a SVM in terms of shape. The training time for AdaBoost is strongly dependent on the number of estimators used and of course, the training data size and dimensions. Further information on the algorithm as well as detailed formulas are presented in [22].

3.3.1.3 K-Nearest-Neighbor

The K-Nearest Neighbor (kNN) algorithm is a comparably simple classifier. The basic idea here is to determine, to which class a vector is closer to. In the feature space, every training vector will be populated with the related class. When a new point should be calculated, the vector is populated in the space, and the distance to the nearest points are measured. The K-Value will determine, how many other vectors will be included in the evaluation. This algorithm resolves, how many of the K neighbors of the vector are from a certain class. The algorithm can be configured with several parameters in order to configure the class decision. Options like the distance measurement method or the influence and amount of data-points next to the predicted vector can be set. The class decision is then based on the most neighbors, which have the smallest distance to the data-point. Depending on the chosen K-value, the decision boundaries can change. An example of this is shown in Figure 3.9.

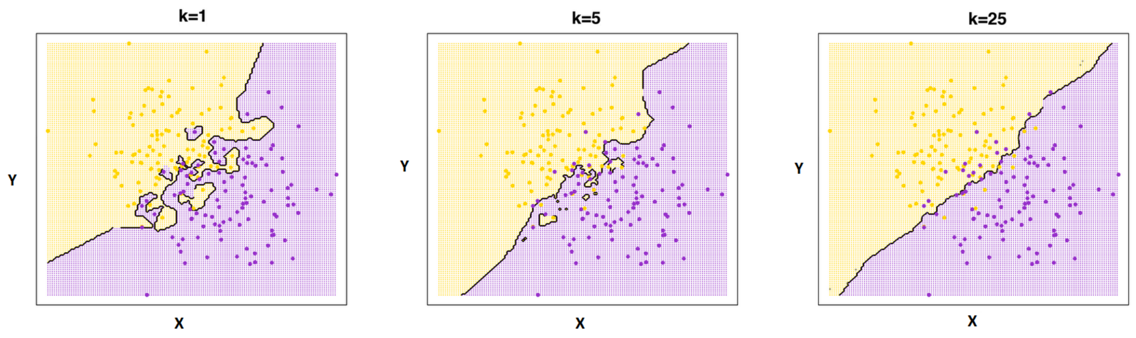

Figure 3.9: kNN decision boundary with different K values

The kNN algorithm is simple but the performance is also very dependent on the feature space and parameters used. There is nearly no training time required, since the feature space of the training data is used to predict the model. The downside of this is, that predictions take a long time, since every instance has to be populated in the feature space and then evaluated based on the algorithm. Also, the size of the training model is as big as the training size. In the case of the full KDD Cup 1999 Dataset, the kNN-Training model would have more than 700 Mb. Further reading for this topic can be found in [26].

3.3.1.4 Decision Tree

The decision tree algorithm is a relatively fast and robust method which uses a tree as a predictive model in order to do classification decisions. It represents leaves as class labels, branches as conjunctions and works similar to a flow chart. To go through this flow-chart like structure, tests are made at each node and based on the given value, decisions are made. After each decision, the next test will be issued until the end of the branch is reached. The last decision leads to a leaf, which determines the class. The construction of this tree is realized by using class-labeled training tuples. Each tuple can be derived into an attribute test, which will be attached to the tree. On the end of each branch, the leaf is represented as the decision for a class. Since the decisions are straight forward, the decision boundary is angular. Decision trees with a small deepness, which means that only a few tests are issued until a class decision is reached, can also be used as a function in boosting algorithms. An example of a decision-tree boundary is shown in Figure 3.10

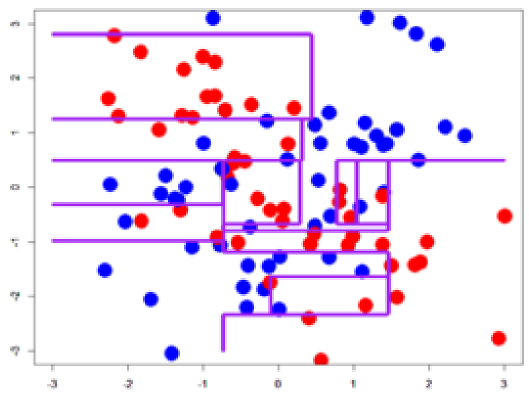

Figure 3.10: Boundary example of a decision-tree algorithm

Depending on the complexity and the deepness of the tree, even boundaries like shown with AdaBoost in Figure 3.8 can be achieved. Decision tree algorithms can be configured with options like deepness-levels and the amount of leaf nodes, in order to control the complexity of the tree. Generally, this algorithm is trained comparably quickly, does not need much data-preparation and also provides predictions very fast. In the same time, depending on the parameters, the classifier can be provide good performance and accuracy. Further information can be found in [33, Ch. 3].

3.3.2 Voting Techniques

As stated in Section 3.3.1.2, voting techniques are already known in pattern recognition. The idea is to classify a vector and exposing it to different decision boundaries. Afterwards, every class-decision of each model counts as a vote for the final decision. There are several methods on how voting can be realized, for example with differently trained models or with variation of the training data. The difference with this work is, that not only weak learners or one differently trained models are used in order to form a voting scenario, but different learning approaches and algorithms contribute to the voting mechanism. The work of Thomas G. Dietterich [9] states many different examples on how voting can be applied in machine learning. It is also argued, that different approaches on how data is classified have each its own flaws and benefits, which might be used in order to create a more powerful classifier with voting. Also, a work of Gareth James [16] uses majority votes to solve classification problems. The basic principle, why a vote can be more efficient than only one classification mechanism, is illustrated in Figure 3.11. In this example, the detection rates of different attack types with a set of classifiers is shown. For simplicity, the scenario only contains false negative cases (grey area) and normal traffic is always classified correctly. Thus, non-attack instances are left out in the illustration.

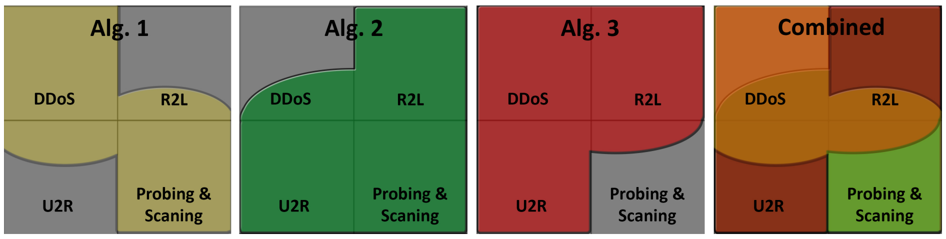

Figure 3.11: Example of an optimal vote scenario

the first three boxes from the left show three algorithms, which have different prediction accuracy for each intrusion type. If an area is grey, the algorithm can not detect this attack type very efficiently and will very likely classify it as normal traffic. As it can be observed, Algorithm 1 can predict DoS attacks as well as probing and scanning, but has very bad prediction for R2L and U2R attacks. Algorithm 2 can't predict DoS and algorithm 3 is weak on Probing and Scanning. When all these models are combined, as shown on the far right, each field has at least two classifiers, which will vote for an attack. This will result in very high accuracy for the predictions, if a majority vote is chosen. However, this is an optimal scenario and is most likely not reproducible in reality. The majority votes can also have a bad impact on the classifier performance, for example if a weak algorithm is selected. There is also the possibility of using a weighted majority vote. Instead of only counting the classifiers which voted for a class, the vote weights vary with each model. This means that the vote decision is done by the amount of weight assigned, and not by the highest number of votes. As shown in several research papers, similar approaches have been applied successfully in other fields [7, 39]. The aim of this thesis is to achieve better performances with a multi-classifier vote than with a single classifier, using the NSL-KDD Dataset. The focus is hereby on false positives. In order to do this, the technique of weighted majority voting is applied.

4 Creation of a Voting-Classification system

In this chapter, the methods and the realization of a voting-classifier is documented, to give an comprehensible and reproducible setup for the voting classifier. In order to create and test an anomaly IDS with the KDD-NSL dataset, several constraints exist and decisions had to be made in order to be able to implement and test a classification system. A set of different libraries and scripts were developed for this purpose. The following chapter presents essential development steps and explains the reasons for some of the test-architecture decisions. The full source code will be provided as a download on Bitbucket.

4.1 Implementation and approach

Several software solutions and libraries exist in order to test and implement classification systems. One very popular machine learning and data mining software is WEKA [31]. This open source tool is a collection of machine learning algorithms and also provides a user-interface, in order to test different classifiers. In addition to the very wide range of provided functions within WEKA, it offers the possibility to create, plot and modify datasets. Since the program is open source and provides interfaces to add new modules, developers can write new algorithms and use the UI for testing. Unfortunately, several Internet sources stated that the source code is not always

comprehensible or completely reliable. In addition to that, personal experiences with developing for WEKA showed a rather fragmentary documentation. As a resulting conclusion, the source code production seemed to be rather difficult and not advised for this type of project. Nevertheless, WEKA was used in this work for simple tests, to check classification results and to carry out several dataset preparation tasks. Additional code development was not required in order to perform these tasks.
To find possible alternatives and to determine the best solution to implement the classifiers, several libraries were collected and compared. A set of possible candidates including Python, .Net and Java-Libraries were gathered. The decision was made in favor for the python library Scikit-Learn, which offers a well maintained documentation and a large set of algorithms. Other useful classification and preprocessing tools are also provided.
To develop python scripts and libraries, several software modules had to be installed. The following list presents the versions and included libraries which were used in the project:

1. Python 3.4.3
2. Numpy 1.9.2
3. SciPy 0.15.1
4. Scikit-Learn 0.16.0

Python 3.0 was released in 2008 and is a new version of the language. It is incompatible with 2.x releases, but the language is mostly the same. Version 3.X differs in many details and functions, for example how built-in objects, like strings, work. Therefore all developed source code will not be executable with 2.x versions. Although Python 2 is still very popular, version 3 was chosen because newer functions, libraries and updates will only be provided for the most recent release.
Numpy is a fundamental package for scientific programming and contains tools to integrate modules written in other languages like C. Among others, it offers linear algebra functions and N-dimensional array objects. SciPy is also a library for scientific and technical computing, since it contains a large set of algebra, plotting and processing

modules. Both packages SciPy and Numpy are mandatory in order to use the Scikit-Learn library.
Scikit-Learn is built on the above mentioned libraries and is a simple and efficient tool for data mining and data analysis. The range of functions includes classification- ,regression- and clustering algorithms, as well as methods for dimensionality reduction and preprocessing. In addition to that, model comparison and validation functions are provided. Further information on this tool can be found in [38].
For developing scripts and libraries, the Python IDE PyCharm professional 4.04 was used.

4.1.1 Dataset preparation and modifications

The KDD-NSL Dataset generally comes in an .arff format but it can also be downloaded as a .txt file. The .arff Format is WEKA-Specific and definitions of attributes and value types as well as ranges are included in this file. In general, datasets can be seen as N times M matrices. Each row N represents one instance of the data. The M columns represent the amount of features, which every instance contains. In the .arff file, the name of each attribute is stated in the beginning, as well as every possible value for the feature. As stated in Section 3.2, some features of the NSL-Dataset are in a string format. For example, 70 distinct string-values are available for the “service” feature. Since a string type is not practical to use for mathematical operations, like they will be executed in the pattern recognition process, every string-value has to be replaced with a distinct number. To do this and other operations, like enabling to use the data with scikit-learn functions, the library “KDDTools” was created.
The very first step was to remove the first 44 lines in each of the training and testing sets of the .arff file, since those are only required when used with WEKA. This process was done manually. After this operation, the data is represented as a comma-separated matrix with each 42 features per row.

Transformation and mapping of NSL-Data

In order to transform the text into a python object and map the data afterwards, the function “genFromNSL” is used. In that function, a file is read and parsed into an object. To do this, Numpy has a method “generatefromtxt” in its package. The comma can be specified as a separator, which then is used to copy the data into a multidimensional Numpy array. Since there is no common datatype for every feature in the NSL set, the strings are saved as byte-strings and numeric values are saved as floats. In the next step, the byte strings have to be decoded and replaced with a distinct numeric value. Also, the last feature, which represents the class, has to be detached from the data. The reason why this has to be done is, that Scikit requires a separate array of the matching classes for each data-point, called targets, in order to train the classifier. The class feature should not be in the training data at any time, since the class attribute will be used over the targets array. Each instance of data and its matching target can be regained by using the same index number. Therefore it is crucial that the data arrangement of both data and targets is not scrambled. If there is a randomization, each column in the arrays are exchanged in the same way to keep the correct matches. The function “KDD_NSLmapper” has been developed in order to create a matrix of data-instances and to generate a separate targets array. It can be found in the library “KDDTools”. The operation that is carried out by this mapper function, is the replacement of strings with a numeric value. The optional Boolean parameter “attack_types” allows to map the classes to the four specified categories. For the voting-classifier, only a two-class pattern recognition problem will be used. That means that the classifier has to distinguish between normal and anomalous traffic. Nevertheless, the attack type mapping is required in order to create derivations of the sets. When the mapper function is called the byte-strings are decoded and replaced with a number. The code sample below shows this approach for the protocol type feature with the value ’udp’.

```
protocol = nsl_Data[i][1].decode('UTF—8')

if protocol == 'udp':

protocol_type.append(2)
```

The variable “i” iterates through every instance and the “1” stands for the column-position of the feature. With that method, every string-value is exchanged and stored in a separate array. In this case, the array is called “protocol_type” and ’udp’ is represented by the number “2”. Before the return of the function is called, the data is put back together and the output is a N times 41 dimensional matrix, consisting of only numeric values. The same method is used to map the classes-feature. Since this feature is either “normal” or “anomaly”, a 0 or 1 value replaces the string to generate the targets array. The mapper function returns both the mapped targets and data instances, in the form of an Numpy array. This is required for Scikit-Learn functions.

Applied Scaling Methods

The next step before the data is used for training and testing, is scaling. This is used to standardize the range of the features and is also known as data normalization. In general, the values, which can for example vary in between 0 and 5000, are scaled into a defined range. Usually this is from 0 to 1 or -1 to 1. When Arcustangens scaling is used, the data is standardized first and afterwards, an arctangent function is applied. The standardization implies that the mean value of the specific feature is subtracted from each number in the matching feature-column. This causes values near the mean to be close to zero. After that, the value of each feature is put in the Arctangent function and multiplied with 2 divided by pi. This puts the data in the interval -1 and 1. The operation causes exceptions of the mean value to be placed on the edges of the interval. The arctangent scaling is usually a very efficient way to scale the data, since it causes non-regular values to be more recognizable.
For general classification tasks, the library “ClfTools” was developed. It contains the methods “scaleAtan” and “scaleSimple”. Since scaling can have a great impact on classification performance as well as training and prediction time, several scenarios are evaluated. For simple scaling applications, Scikit offers various functions which are used in this library. Arctangent scaling does not exist in the Scikit module and has therefore been programmed. The Listing below shows an extract of the scaling function.

```
def scaleAtan(trainData, testData):
```

```
std_scale = preprocessing.StandardScaler(copy=True)
std_scale.fit(trainData)
trainData = std_scale.fit_transform(trainData)
testData = std_scale.fit_transform(testData)
trainData = 2 / np.pi * np.arctan(trainData)
```

First, the “StandardScaler” from the Scikit module “preprocessing” has to be instanced. After that, the scaling object is fitted to the training data. This means that several values, like the means, are calculated and stored for further use. This is crucial for classification, since both training and testing data have to be scaled with the same values. Otherwise, the results are distorted. In the next step, both training and testing data are scaled with the function “fit_transform”. This method scales the data according to the calculated values, which were determined from the data given in the “fit” function. In general, this is always the training data. The last operation is to use the Arctangent and multiply it with 2 divided by pi. In this case, mathematical values and operations are used from the Numpy module “np”. It has to be mentioned that if an array is given to the mathematical operation, as it is stated in the last line of the listing, the operation is taken out on each of the array elements. A loop for iteration is not required in this case.
The tests with the classifiers are carried out with three different preprocessing approaches. Unscaled data, normalized data in between the range of -1/1 and Arctangent scaled data is used. It also has to be mentioned that scaling can have a large impact on the training time, based on the applied algorithm. Especially the SVM-Model may require about five times longer for training, when no scaling method is applied.

Saving and Loading Prepared Data

The “KDDTools” library also contains several other functions to ease the use of the NSL-Dataset. Since mapping and preprocessing of the data takes time and resources, it is unpractical to do this process for every test situation. It also has to be mentioned, that the data does not change, unless another scaling method is applied. Therefore the functions “saveDataset” and “loadDataset” were defined. These enable to save

and load Numpy objects, which is considerably faster than doing the whole preprocessing. Another advantage is, that targets and data are stored in separate objects, which makes it more applicable for Scikit-learn.
Another interesting factor for performance analysis is, which and how many of the different attack types were found by a classifier. Considering that only a two-class problem is executed with the classification, the method "findIntrusionType" was defined. With this function it is possible to find which intrusions of a certain type were found. This works only when the regular and unmodified NSL-KDD test or training data is used. The function loads a stored array of the intrusion types, which occur in the specified set, and then uses the given targets array to exchange the binary values with a five-class detection. Simply put, each detection will be exchanged with its matching intrusion type. This can only be applied when the classification data is still in the same order.
The last important function for dataset preparation is the "makeDataset" method. It will be used to create datasets containing only specific attack types. The listing below shows the function header.

```
def makeDataset(data,targets,classes):
```

When a dataset with multiple classes is given to the function, the "classes" parameter is an array that specifies, which of the different targets should be put in the resulting set. Since the KDD-NSL dataset can be downloaded with the exact intrusion types, it is possible to map this data to the more sophisticated five-class problem, consisting of normal traffic, R2L, U2R, Probing and DoS attacks. With the "makeDataset" function it is possible to create a training or testing set, which only contains the specified classes.

Script for Dataset Creation

To enable easier and faster testing, it is practical to use a script to generate the datasets. Like this, the data can be mapped, preprocessed and scaled. Afterwards, the object can be saved to a specified location. If a change of the scaling method is done, the new sets can be saved and reused in the same way. This facilitates test scenarios and makes changes to datasets more easy. Because of these advantages, the script

"CreateNSLSets" was developed. In general, the execution of the script does three steps. The first is to load the data, transform it into a Numpy object and map the string values to numbers. After that, the data is scaled in the specified way. To be able to reuse the data in testing scripts, the targets as well as the data is then stored in a given location. If the file already exists, it will be exchanged against the newer version. Another task that is executed with the Script is the generation of a validation dataset. The use of this set is to check, weather the classification is not fitted to the testing set, but also is able to classify "unseen" data with roughly the same performance. With this approach, overfitting for the training and testing set is avoided. In Section 4.1.2, further reading and details on the validation-set construction and architecture will be given.

4.1.2 Testing of Classifiers

When combining algorithms, it has to be ensured, that every component works correctly. For this purpose, tests of each classifier were made before they were used in the vote scenario. The following sections will describe which parameters were chosen, how tests have been carried out and which results were achieved.
To train the classifiers, the datasets are be generated with the script "CreateNSLSets" and then loaded with "loadDataset". Before every classifier can be created, the appropriate module has to be imported and assigned to a variable. In addition to that, there is a possibility to modify the default values of each classifier. As it is stated in Section 3.1, every algorithm can use different values and attributes in order to calculate a decision boundary. Those values are given when the objects are instantiated. This process is shown in the listing below.

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
dTree = DecisionTreeClassifier(max_depth=None,class_weight='auto',min_samples_leaf=20),
aBoost = AdaBoostClassifier(DecisionTreeClassifier(max_depth=3),n_estimators=70),
kNN = KNeighborsClassifier(n_neighbors=1,weights='distance',algorithm='kd_tree',leaf_size=50),

svm = SVC(C=1.0, kernel='rbf', cache_size=500, verbose=True)]
```

These algorithm parameters are used for training with the NSL-KDD dataset. The Decision Tree classifier is configured to have at least 20 leaves. The AdaBoost classifier also uses a Decision Tree as base estimator, but the maximum depth is restricted to three levels. This allows the classifier a have simple and fast learning algorithm. In addition to that, only 70 estimators are given, but if less are required, the algorithm will stop before. These two classifiers have quite similar behavior and training, but the AdaBoost-Classifier will add a more complex decision boundary with its ensemble voting technique. Very different from that is the Nearest-Neighbor classifier, which is only configured to use one Neighbor. Therefore only the closest neighbor is chosen. A regular distance measure is configured to determine the closest instance. This means every data-point close to an intrusion-point, will be classified as such. The same works the other way round. The last classifier is the SVM, which is configured to use a Radial Basis Function. The C-Parameter configures the optimization, which implies the tolerance for allowed misclassification in the training set. A low value will result in a rather loose boundary and will avoid overfitting. This low value is used since the testing set contains a lot of unknown data and attack types. The cache size and verbose parameters are only for computation and output purposes. It has to be said that the SVC-Module of Scikit is an adaption of the libSVM library. This is due to the problem that libSVM does not use the same commands for fitting and prediction. The SVC-module therefore simplifies this library and adapted the methods. Nevertheless, it uses the libSVM source code in the background. After the Instantiation and parameter configuration, the models have to be trained with the training set. This is carried out for each Scikit classifier with the method “fit”. For supervised classifiers, as it is the case here, the targets array with the matching classes always have to be given for training. When the model is trained, the classification can be done with the method “predict”. This will trigger the classifier to estimate each data-point of a given array and apply its trained and calculated decision boundary. Naturally, the correct class is hereby not known and will not be given. The output of the predict function is an array with the predicted targets of each data-point. This also implies that each instance can be regained by its index. An example of this process is given in the listing below.

```
svm.fit(trainData, trainTargets)

svm_out = svm.predict(testData)
```

Result evaluation method

Since the result of each prediction has the same size and order as the correct targets array, a comparison between the output can be executed in an easy way. One approach can be to compare, how often the results from targets and predicted array differ. Nevertheless, this would only give information about the general precision. No knowledge about FP,FN,TP and TN rates can be gained from this method. Therefore a confusion matrix can help to clarify the classification performance. In the matrix, each row resembles a class. Row zero and column zero represent the class “normal”, since it has been mapped this way. Class one is represented in class and column one, which is the case if an attack prediction is carried out. Each match of “zero” elements in the targets- and predicted array will cause to add up the number in the zero-zero position of the matrix. The same principle is applied when a match of “one” elements is found. In case a class is classified incorrectly, it will be added to the “wrong” column. To illustrate this, an example with the matching indicator is shown in Table 4.1

Confusion Matrix		
Class 0	5000 (TN)	500 (FP)
Class 1	100 (FN)	3000 (TP)
	Class 0	Class 1

Table 4.1: Example of a confusion matrix

This matrix can also be extended to more classes and is therefore used to give distinguished results for detection of the different attack types. To generate a more readable output and develop several functions for classification problems, the library “ClfTools” was created. Among others, this library contains the function “printStats”, which prints the amount of correctly and incorrectly classified data, calculates percentages and gives an overall accuracy as well. For this purpose, the Scikit module “metrics” is used to generate an confusion plot. After that the formulas for each classification type are applied and printed to the console. The table 4.2 gives the classification results for each of the classifiers, when using the default KDD-NSL training and testing sets.

Classification Results				
	Dec. Tree	AdaBoost	kNN	SVM
TP	85%	91%	97%	96%
TN	78%	76%	68%	70%
FP	15%	9%	3%	4%
FN	22%	24%	32%	30%
ACC	81.2%	83.1%	79%	80%

Table 4.2: Classification results of each classifier with NSL-KDD training set

Naturally, false classifications should have low rates and true classifications high rates. As it can be observed, Decision Tree and AdaBoost have best accuracy because these classifiers find more normal traffic instances, called true negatives (TN), than SVM and kNN. With that comes an decreasing FN rate but the FP-Rate increases. SVM and kNN can find many attacks but are weak with finding normal traffic. Therefore, fewer FP classifications occur. This behavior can be used in advantage for a voting system.

4.1.3 Combination Technique

To combine the classifiers, a weighted voting technique was chosen. This implies that for every vote, a certain weight is assigned, based on the classifier performance. To achieve the best results, different weights will be used when a classifier either votes for an attack or against it. The calculation of this weight will be presented in Section 4.1.3. To carry out this voting and weight checking, each classifier has first to be trained and the predicted output has to be saved. Every weight is then added up to a two dimensional array. The array used for storing the vote weights is shown in the listing below.

```
weights = []

for j in range(0,len(testTargets)):

weights.append(np.array([0,0]).astype(float))
```

The code creates an object similar to an N by two matrix of weights. The first index represents the position of each vote. This means each weight of a data instance can be regained by its index. The created matrix can be seen as an more advanced, two-dimensional targets array. In the first column is used to store the weight for a “normal”-vote, the second is for attacks. Afterwards, this array will sum up votes of each classifier. This process is shown in the following listing:

```
for prediction in predictions[i]:
if prediction == 0:
weights[j][0] = weights[j][0] + model_weights_normal[i]
else:
weights[j][1] = weights[j][1] + model_weights_attack[i]
j=j+1
```

Each model has the predicted array of targets, called “predictions”. With a for-each loop, every vote, which can be either zero or one, is picked up and weights are summed up for the matching column. As it can be observed, the weights can be different, since two independent arrays, containing the vote weights, are used to sum up the array. The “i” variable represents an iteration through each model. The “j” variable is required to go through every index and assign the weight. After this process, the final vote-classifier prediction is carried out as stated below:

```
for x in range(0,len(testTargets)):
if weights[x][0] >= weights[x][1]:
vote_clf.append(0)
else:
vote_clf.append(1)
```

In this case, an iteration in the length of the targets is carried out and the weights of each vote are compared. Based on which number is higher, the vote is assigned to have an attack or normal traffic prediction. If the weights are the same, the decision is in favor of the “normal” classification.

Weight Balancing Function

In order to assign and calculate weights to each model, a function was developed in order to balance the given amount in an efficient way. Due to the problem that several classifiers might work better than others, it is crucial to assign more or less weight, based on the performance. To do this, the following function $f_w(x)$ (4.1) was developed:

$$f_w(x) = \frac{1}{(1-x) * A + B} \tag{4.1}$$

The variables A and B are used for adjustments. The Variable X stands for the precision of TP or TN values. Those values are typically between 0 and 1 and represent the probability of a correct prediction of either normal traffic or an attack. The closer the value is to 1, the better is the prediction rate. This means that high values should produce high weights, which is enforced by the formula. Weight values increase strongly as they get closer to one, on the other hand, lower values in classification performance will be penalized with low weight. The A value is used to control the slope of the function. A small A value will create a slower rising slope, beginning with lower x values. This means that even low performance numbers get a higher base weight and the impact of the formula is decreased. On the other hand, when higher values are used,only very well performing classifiers will have higher weights assigned. The maximum achievable weight is also decreased. To be able to control the highest assigned value and to avoid an infinite number for the weight, the B value can be adjusted. Very small B values will allow the weight value to rise very high, when x is close to 1, smaller B values will decrease the slope and also decrease the impact of the formula once again. For example, if a 1 is chosen for B, the weight formula is practically not existent and therefore, the classification can be compared to a majority vote. Nevertheless, this approach allows to balance different weights for each TN or TP values, which can also push the results to either higher FN or FP rates. The sample values which are used in the voting scenario, as well as a graph for this formula, will follow in the next section.

4.1.4 Experimental setup

To do a test with the a vote-classifier, several steps have to be done as follows:

1. Instantiation and training of classifiers
2. Carry out a prediction of each classifier with a small data set
3. Comparison and decision for a vote, based on the assigned weights

The small data in step 2. is required in order to calibrate weights, since they are calculated based on the classification performance. The first and the last step are already described in the last chapters. Therefore this section will further describe how the weights are assigned and which data was used to carry out the calibration.
Two approaches are possible in order to get data for weight calibration. Either a split of the training set can be used, or a part of the training data. Since the regular testing data of the NSL-dataset is very difficult to predict, a 50% Split of the NSL-KDD test-set was created to calibrate the weights in a first step. To split data and generate two new datasets, the function “randomSampling” in the library “ClfTools” is used. This function randomly scrambles the data and returns two datasets with matching targets. The length of the sets are defined with the split-size value.
In addition to that, another method was developed and carried out in order to test the vote-classification performance. The method differs, because the whole KDD-NSL dataset is used and training and testing sets are merged. Afterwards, the set is randomized and split into a training- calibration- and testing set.
To calibrate the weights and to keep things simple, the value 1 was used for the constant A. The B value was set to 0.01, to allow a maximum weight of 100. The result of this is shown in formula 4.2.

$$f_w(x) = \frac{1}{(1-x) + 0.01} \tag{4.2}$$

Naturally, more differentiated calibrations with the formula can be done, which will be discussed in Section 4.1.6. To further illustrate the selected function, the graph in

Figure 4.1 shows the impact on the x-values.

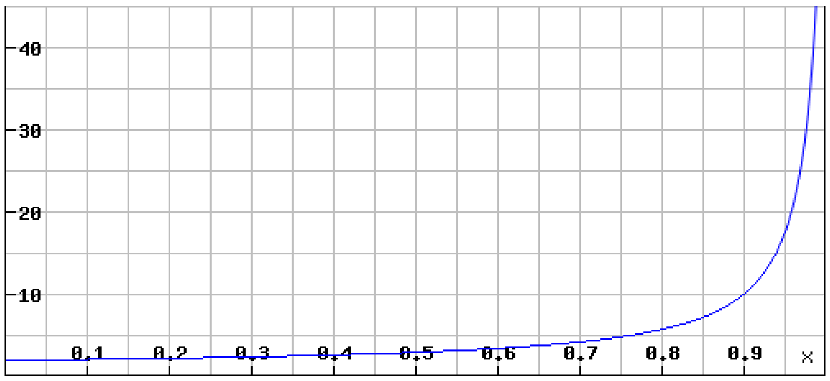

Figure 4.1: Graph of the weight balancing function

The graph shows, how TP- or TN-values with a correct prediction rate over 0.9 will generate strongly increasing vote weights. After the weights have been calculated with a calibration set, the vote-classification will be carried out with a test-set.

4.1.5 Results

In order to test the weight balancing function of the vote-classifier, several scenarios were created and tests were carried out with either a modified or unmodified NSL-KDD dataset. Every test is presented in a table, with the classification performance of each classifier and its TN,TP,FN and FP rates. Each test generates two results. The first table presents the vote classification with calibrated weights from the same test set. Afterwards, the same models and the calibrated vote-classifier is used with a verification set. The parameters of the classifiers were never changed, only the training and testing data was exchanged. The first test was carried out with Arctangent scaling and with the regular KDD-NSL dataset. Table 4.3 and 4.4 present the results for the test and the verification set.

This table shows the classification results and the calculated weights after testing with a 50% split of the regular NSL-KDD test set. As it can be observed, the large amount of unknown data impacts on the classification performance. This is noticeable due to the high false negative rates, which represent undetected intrusions. It implies that most of the classifiers evaluate 20 to 30% of the roughly 6000 intrusions as normal traffic. In addition to that, the FP rates are also rather high, which occurs especially

Classification results & calculated weights					
Type	Decision Tree	AdaBoost	K-Neighbors	SVM	Vote
TP	84.89%	90.58%	96.38%	95.83%	91.03%
TN	77.42%	75.71%	68.59%	70.10%	80.34%
FP	15.10%	9.41%	3.61%	4.16%	8.96%
FN	22.57%	24.28%	31.40%	29.89%	19.65%
ACC	81.50%	82.95%	79.36%	80.51%	85.88%
f(w)	5.8 / 4.2	9 / 3.8	20 / 3	16.6 / 3.8	TP / TN

Table 4.3: Classification results for a 50% split of the NSL-KDD test-set with A-tan scaling

in the better performing classifiers like AdaBoost and Decision Tree. This implies that a trade-off with correct TN and FP rates occurs. Either the FP rate is low and less normal traffic is classified correctly, or more normal instances are found and the FP rate is higher. Considering the assigned weights, SVM and K-Neighbors have best rates in finding attacks, but with a high FN trade-off. This also results in lower weights for TN values. Decision Tree and AdaBoost perform better in finding normal traffic but are weaker with attacks. Combined in a voting scenario, the overall detection rate can be increase by roughly 3%, which achieves an overall of 3300 more correct predictions compared to the best single classifier performance.
Of course, the weights have only been calibrated here, which means that the classification performance after the calculation has also to be tested on new data. This will be done by using the verification set. The results of this test is shown in Table 4.4.

Classification Results for verification set					
Type	Decision Tree	AdaBoost	K-Neighbors	SVM	Vote
TP	85.82%	91.25%	96.90%	96.38%	91.77%
TN	77.64%	68.59%	67.46%	69.38%	79.81%
FP	14.17%	8.75%	3.09%	3.61%	8.22%
FN	22.35%	24.19%	32.53%	30.61%	20.18%
ACC	82.14%	83.41%	78.77%	80.32%	87.01%

Table 4.4: Classification results for a 50% verification-split of the NSL-KDD test-set with A-tan scaling

For the verification set, the classification performance only varies in small percentages between 0% and 2%. This is crucial to the weight-balancing process, since the classification performance is also expected to be the same with new data. It also implies that

the data has to be about the same and should not generate completely new performance values for the classifiers, otherwise this method can not be applied efficiently. In this case, the vote-classification can boost the accuracy rate up to 3.5%, compared to the best model. Even though this might not seem much, considering the low detection rates, it can be interpreted as a large increase. Another reason is that during research, the author was not able to find any reproducible classifications with original NSL-KDD data, which have performed better than 82% accuracy. Another observation is, that FP rates are lowered compared to the best classifier, but the lowest FP-Rates of other classifiers can not be achieved.
For further performance analysis, a confusion plot for the different attack types was generated. The chart presented in Figure 4.2 shows the existing and detected intrusions by the vote classifier.

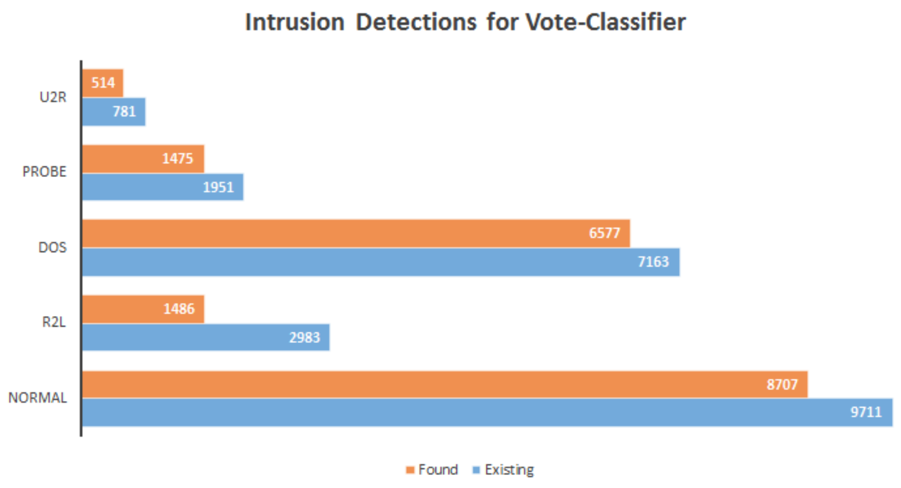

Figure 4.2: Chart of detected and existing intrusions with vote classification

The results demonstrate, that a large gap of detected and existing R2L attacks occurs. This is most likely due to the lack of R2L instances in the training data. Other attacks can be found with better accuracy. About the same ratios were produced with other classifiers but always with less accuracy. Significant variations of detection rates, when applying other algorithms, were not observed.
In the next scenario, the training as well as the classification will be carried out with only normalizing the dataset and scaling it in the range between 1 and -1. This means that outliers are not placed as striking as in Arctangent scaling. The results of the calibration dataset is presented in Table 4.5.

Classification Results for test-set & calculated weights					
	Decision Tree	AdaBoost	K-Neighbors	SVM	Vote
TP	90.40%	96.32%	97.39%	96.68%	90.93%
TN	66.33%	63.93%	67.99%	65.54%	73.70%
FP	9.59%	3.67%	2.60%	3.32%	9.06%
FN	33.66%	36.06%	32.00%	34.45%	26.29%
ACC	76.12%	75.05%	79.11%	76.69%	81.82%
f(w)	9 / 2.8	20 / 2.6	25 / 2.9	20 / 2.7	TP / TN

Table 4.5: Classification results for a 50% split of the NSL-KDD test-set with normalized scaling

It is not surprising, that the classification results have less accuracy than before. Nevertheless, the vote-classifier can prove its efficiency. In this case, an increase of about 3% can be observed within the calibration set. A larger change in performance rates occurred with the switch of the scaling method. Only the K-Neighbors algorithm could roughly keep the performance. Other models had a dropping accuracy and AdaBoost as well as the Decision Tree are now less efficient than the SVM. This is completely opposing to the results with arctangent scaling. With normalized scaling, more attacks are found in general, but false negatives are increased more drastically. If a classifier votes for a normal behavior in this case, it is quickly outweighed with an attack-weight, which is comparably high for each classifier. Table 4.6 shows the results for the verification set.

Classification results for verification-set					
	Decision Tree	AdaBoost	K-Neighbors	SVM	Vote
TP	90.12%	96.25%	97.42%	96.87%	90.72%
TN	64.95%	62.94%	67.06%	65.06%	72.66%
FP	9.87%	3.74%	2.57%	3.12%	9.27%
FN	35.05%	37.05%	32.93%	34.93%	27.33%
ACC	75.03%	74.22%	78.45%	76.45%	81.14%

Table 4.6: Classification results for a 50% verification-split of the NSL-KDD test-set with normalized scaling

In this scenario, the results did not vary in a large scale, only the accuracy rate dropped by about 1% in most of the models. However, the vote-classifier was able to improve the performance again with roughly 3%.

Another scenario was tested with completely unscaled data. Again, the same process with first calibrating the weights was used, as stated in the other scenarios before. After calibration, the verification results, as they can be found in Table 4.7 were produced.

Classification results for verification set					
	Decision Tree	AdaBoost	K-Neighbors	SVM	Vote
TP	91.81%	96.35%	95.94%	96.63%	92.08%
TN	65.12%	66.95%	66.18%	59.51%	71.01%
FP	8.18%	3.648%	4.05%	3.36%	7.91%
FN	34.87%	33.04%	33.81%	40.48%	28.98%
ACC	75.45%	78.03%	77.24%	70.37%	80.34%

Table 4.7: Classification results for a 50% verification-split of the NSL-KDD test-set without scaling

It can be observed that the classification results show even less performance than before and the vote classifier is able to boost the performance by about 2%. Other than the quite stable accuracy improvements, this example also shows that scaling impacts the classification performance positively, when the regular NSL-KDD is applied. With a scenario of several classifiers, which are able to perform in a range of 70% to 84%, the vote-classifier seemed to work well.

Results with a Modified KDD-NSL Dataset

To generate more different results for performance comparisons, a scenario setup with larger accuracy gaps and also testing with easier data might be useful. Therefore, the NSL-KDD test and training set were assembled to one dataset. This results in a large set with all possible attacks and normal traffic instances. After that, the instances are scrambled and split, which is the usual process when pattern recognition is carried out. The final result is a test set with less zero-day attacks and a training set with more different intrusion types. These changes lead to a higher accuracy and different classification results. To use this setup with the Vote-Classifier, the data was prepared as stated in Section 4.1.4. 80% of the full data were used for training, 15% for calibrating the votes and 5% for verifying the vote-classifier. The following examples will only show the results for the verification set. It has to be mentioned that due to the randomization of the data, which is carried out each time the sets are generated, the classification results may vary. Table 4.8 shows the achieved performance for an Arctangent scaled verification set.

Classification Results for verification-set complete set,A-tan scaling					
	Decision Tree	AdaBoost	K-Neighbors	SVM	Vote
TP	73.34%	90.75%	99.79%	98.65%	99.88%
TN	99.53%	99.86%	99.82%	97.81%	99.72%
FP	26.65%	9.24%	0.20%	1.34%	0.11%
FN	0.46%	0.13%	0.17%	2.18%	0.27%
ACC	83.24 %	95.27%	99.81%	98.19%	99.79%

Table 4.8: Classification results for a 5% verification-split of the full NSL-KDD set with A-tan scaling

In this case, the classification results show large variations. The Decision Tree has low performance in terms of finding attacks but has a good normal traffic detection. This will give the classifier a high weight for the matching class. Also, AdaBoost seems to have the same detection issue, which is most likely due to the similar algorithm. This problem also may be due to the Arctangent tan scaling, which can result in a more complex feature space. It might be required to tune the parameters of the classifiers Decision Tree and AdaBoost in order to get better results, for example by allowing more estimators. Especially the K-Neighbors algorithm performs very well here and a large performance gap is created. For the Vote-Classifier, the overall accuracy is just 0.02% less than that of the K-Neighbors. So in this case, the overall performance can not be improved, but the FP-rate is the best achievable. With only 0.11%, only half of the false detections occur, compared to the best model. Most likely, this is due to the high TN rates of the three classifiers. If this behavior can be reproduced, it would be very attractive for a Smart Grid implementation. In another scenario, the Vote-Classifier proves to be even more useful. Table 4.9 shows the results when normal scaling is used.

Classification results for verification-set					
	Decision Tree	AdaBoost	K-Neighbors	SVM	Vote
TP	99.46%	99.77%	99.65%	98.00%	99.88%
TN	99.48%	99.59%	99.56%	97.90%	99.71%
FP	0.54%	0.23%	0.34%	1.99%	0.11%
FN	0.51%	0.40%	0.43%	2.09%	0.28%
ACC	99.47%	99.67%	99.60%	97.95%	99.80%

Table 4.9: Classification results for a 5% verification-split of the full NSL-KDD set with normalized scaling

Even though this scaling method has not proven to be very efficient in the regular NSL-KDD sets, it changes when using the complete dataset. Arctangent scaling may work in many cases, but also might place data-instances in an unfavorable way. This can be case when the variation in an attribute is very high. In this case, regular scaling might work better, because the changes made to the feature space are a lot less altering. Since the features were developed to place attacks away from regular traffic instances, good results can be achieved when more intrusions exist in training data. Every classifier has now good performance, only the SVM has comparably less accuracy. The best classification can again be achieved by the vote classifier, which performs 0.13% better than every other classifier. Again, this reduces the false positives by more than a half. In addition to this improvement, the vote-classifier also provides the best accuracy and with that, the best FN rates. Even though the limits of classification can not be pushed much further, due to the good results of each classifier, the voting is able to improve the performance. Similar results can also be found in the next scenario without scaling, as Table 4.10 shows.

Classification results for verification-set					
	Decision Tree	AdaBoost	K-Neighbors	SVM	Vote
TP	99.62%	99.88%	99.70%	99.88%	99.88%
TN	99.47%	99.85%	99.8%	97.84%	99.80%
FP	0.38%	0.11%	0.29%	0.11%	0.11%
FN	0.52%	0.15%	0.2%	2.15%	0.20%
ACC	99.54%	99.86%	99.75%	98.76%	99.83%

Table 4.10: Classification results for a 5% verification-split of the full NSL-KDD set without scaling

The best performance now can be achieved with AdaBoost and the Vote-Classifier only has 0.03% less accuracy and the same FP-rates. This result is not surprising, since most of the classifiers have very similar and also a very high performance. This results in getting mostly the same weight for every classifier. Nevertheless, the Vote-Classifier still provides one of the highest efficiencies and is not a considerable downgrade compared to AdaBoost.

For better result comparison, table 4.11 shows best achieved performance for the most accurate model and the voting classifier. It has to be noted that in the cases of NSL/Atan and NSL/None , the best accuracy model does not equal the best FP model. With Atan scaling, the voting-technique was able improve FP rates of the best accuracy model. All other instances have the same model and can be compared directly. Another important fact is, that improvement of accuracy always implies, that either better FN or FP rates were achieved with an overall of more correct detections.

Set/Scaling	Best Acc.	Best FP	Vote-Acc	Vote-FP
NSL/Atan	83.41%	3.09%	87.01% (+3.6%)	8.22% (+5.13%)
NSL/Norm	78.45%	2.57%	81.14% (+2.7%)	9.27% (+6.7%)
NSL/None	78.03%	3.36%	80.34% (+2.31%)	7.91% (+4.55%)
mod/Atan	99.81%	0.20%	99.79% (-0.02%)	0.11% (-0.09%)
mod/Norm	99.67%	0.23%	99.80% (+0.13%)	0.11% (-0.12%)
mod/None	99.86%	0.11%	99.83% (-0.03%)	0.11% (0.00%)

Table 4.11: Achieved performances for vote-classification

The results show, that voting was able to get more accuracy in most of the cases. When this was not achieved, the FP rates were reduced or equal. This implies that voting might help to improve classification results and its implementation can be considered, when the boundaries of a classification system should be pushed. Of course, the performance values might be able to be adjusted to the models in a more efficient way, depending on the applied models. The suggest formula is merely a way to simplify combinations and an approach to get favorable results in most applications. However this was a more general approach, the system was able to significantly improve results in 3 cases and give a slightly better outcome in 5 of 6 cases, by only swapping scaling methods and training sets. An individual adaption to the applied models might bring even more improvement.

4.1.6 Performance Improvement Function

As the tests have shown, the vote-classifier is able to improve the accuracy and often also enhances the false positive values. Nevertheless, the created formula is only an estimation, based on how the weight can be assigned in an efficient way. To further investigate, if the assigned constants A or B of the formula can be improved, an algorithm was developed. The function "findFormulaParams" can be found in the library

"ClfTools" and only requires two parameters. The first parameter is a list of votes from each classifier, the second is an array of correct targets. In a first step, a common range for the values A and B is set. To iterate through each range and test the vote-performance, the function "getBestParams" is called. For testing purposes, it is also possible to use this function with own specified values and a different formula. In this case, the voting scenario is carried out with the preset ranges of the executing function. The best achieved value for the classification is given back afterwards. For A the testing range it is set between 0.1 and 5, with 0.25 steps. For B the range is between 0.1 and 5, with 0.25 steps. When the best value is returned, the algorithm tries to optimize the formula and checks, weather better values can be achieved. This is executed by using more fine granulated ranges, which are closer to the value of the last returned best performance. Based on the available computing resources, used classifiers, specified ranges and the amount of array-length, this process can take long. In summary, the basic idea is to go through each possible slope and value-formation of the formula, as shown in Figure 4.3.

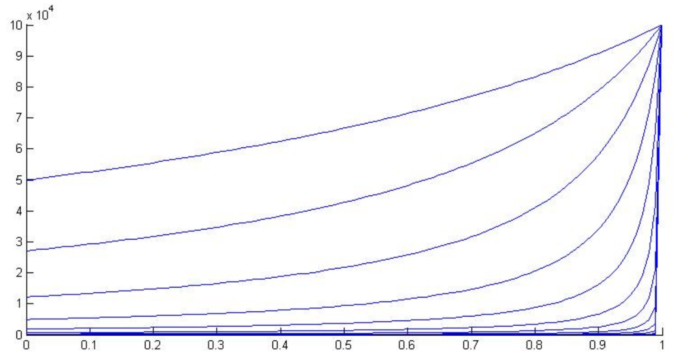

Figure 4.3: Weight function graphs of different parameters

It has to be mentioned that this approach is only experimental, but some test results already showed that vote-results have been improved in several cases. Nevertheless, the improvements were not significantly higher, which means that the current formula seems to be a good estimation. Improvements of about 0.015 to 0.025% compared to the regular vote classification accuracy have been observed. Not only this experiment could determine if the specified formula is close to an optimum, it also enables the exact calculation of the parameters for future use. Since the constraint is to have similar classification performance and a test set for weight calibration is always required, it is useful to apply this technique for best possible results. Naturally, this may only one of many available solutions, but further investigation into other formulas and approaches for weight calculation would reach beyond the scope of this thesis.

4.1.7 Simple voting techniques

When experimenting with voting methods, a few different approaches were elaborated before the finally resulting formula was developed. Several ideas on how voting can also be realized, is presented in this section. The very first option was to use the strongest classifier, if a set of models did not agree on a prediction. This was simply realized by iterating through the votes of each algorithm and compare them with each other. The results of this approach showed an overall increase in accuracy but could not lower FP rates. Therefore, the kNN algorithm, which had the lowest FP-rate in the scenario, was used for the strongest classifier. A code sample of this process is shown in the listing.

```
for i in range(0,len(testTargets)):
if (nn_out[i] == 1):
votes.append(1)
elif (boost_out[i] == 0) and dTree_out[i] == 0) or
(boost_out[i] == 0 and svm_out[i] == 0):
votes.append(0)
elif (boost_out[i] == 1 and dTree_out[i] == 1) or
(boost_out[i] == 1 and svm_out[i] == 1):
votes.append(1)
else:
votes.append(nn_out[i])
```

When the code is executed, the outputs of each classifier are compared. nn_out is the array for the kNN, others are self explanatory. In this case, the kNN decides when an attack is found. Otherwise AdaBoost and Decision Tree, or the SVM, have to agree on a vote. If this is not possible, the decision of the kNN will be used. The stated scenario was tested with the regular KDD-NSL set with Arctangent scaling. In this case, the intention was to reduce the FP rate, but keeping about the same accuracy. AdaBoost achieved a precision of 83.16% and a FP rate of 9.1%. Even though the FP rate is very high, AdaBoost still had the most correct predictions. The kNN algorithm has 79.07% accuracy, but an FP rate of only 3.35%. Therefore, the kNN was chosen to set the attacks, but normal decisions had to be carried out with other models. The resulting vote-classification achieved an accuracy of 84.31% and lowered the FP rate down to 8.62%. This means that overall prediction as well as the FP rate were improved slightly. For further tests, another scenario was developed to achieve a more drastic decrease of the FP rates. To do this, it should be more difficult for the voting mechanism to produce attack-predictions, in order to overrule the kNN algorithm. Therefore, the AdaBoost, Decision Tree and the SVM output had to be the same if an attack prediction is made. This is contrary to the first scenario, which only required two classifiers. The results show a large impact of the small change: The vote-classification achieved only 82.82% accuracy, but FP rates were lowered down to 4.26%.

The tests show, that simple voting scenarios with a primary model and an overrule possibility can also improve classification performances in a desired and controlled way.

4.1.8 False Positive Reduction With Weighted Voting

To apply the FP reduction technique of a simple scenario to the weighted voting, another approach had to be used. To lower or raise the calculated weight based on the FP value, it had to be integrated in the formula. This was carried out by influencing the numerator of the weight function. The updated function is now as stated in formula 4.3.

$$f_w(x, y) = \frac{\frac{1}{y}}{(1-x) + 0.01} \tag{4.3}$$

The x-value is still used for TN or TP values, and the y value represents the FP-rate. This leads to an overall weight increase, when a low FP-value is used. In case a 0.10

FP-rate occurs, the numerator is 10. If a very low FP-rate with only 0.01 is used, the numerator will be increased to 100. Nevertheless, all TN,TP and FP values are important, since the slope of the function will rise strongly with good performances. For the test scenario, the regular KDD-NSL sets were used, and Arctangent scaling was applied. With this setup, the weighted vote classifier achieved 83.03% accuracy and lowered the FP rate to 4.53%. For comparison, the classic weighted vote achieved 86% accuracy, with an FP-rate of 9%. The simplified voting achieved 82.82% and an FP-rate of 4.26%. This means that FP rates were still lowered by over a half compared to the best classifier and the classic weight-technique, but an accuracy of 83% could be kept. Of course, the formula can be further adjusted and the impact of FPs can be increased or lowered when modifying the numerator. Naturally, it is always better to have more correct predictions, even if the FP rate might increase. As stated in the first sections, security breaches in the smart grid can be quite drastic, and therefore it is very important to find as many intrusions as possible.

4.1.9 Observed constraints

Several observations of constraints have been made during the testing phase. These define the limits of creating a successful vote-scenario and therefore are presented in this section. As stated before, one big constraint is the dependency of the classification performance. Once the performance differs in larger ranges, for example when new data is used, the voting mechanism might not work anymore. This is due to the weights, which are balanced based on the estimated classification accuracy. Because of this problem, an additional test case, where vote weights are calibrated with the regular NSL-training set, was not carried out. It is most likely that voting may decrease performance, since the test set has a different structure. In other scenarios, it might be possible to use the training set to conduct the weight calibration. A good application example is the use of K-folds. This way, the performances for each K-fold are determined and the mean values can be used to compute weights.

A second problem which can occur are unequally or baldy trained classifiers. Even though the formula will only use lower weights for classifiers with bad performance, it might influence the classification in an unfavorable way. Naturally, it does not make

sense to use considerably weak classifiers, when better ones are available. When it comes to weak or baldy trained algorithms, it might happen that the classification is very unilateral. For example if most packets are interpreted as an attack. This will create high TP values, since most of the attacks might be discovered. It will also result in a high FP rate, but the suggested weight calculation for the attack performance will not be affected by this occurrence. In the worst case, when every packet is found in the calibration set, this comparably weak model will impact the results in a bad way, with its very high attack-prediction weight. One approach to avoid this is to adapt the formula by lowering the weight, based on the false predictions, or by integrating the overall performance as it is stated in Section 4.1.8. Otherwise it is best to integrate only tested and well performing classifiers. A very efficient classifier is very likely to perform better individually, than in a vote-set with other, considerably weaker models. The third constraint is the amount of algorithms or models used. Smaller tests showed that the best accuracy was achieved with four models. Lesser models can not influence the weight-decision enough to shift the voting results. Four classifiers is the suggested and tested scenario. A set of more classifiers may also perform well, but was not tested due to the higher resource requirements.
A last constraint is dedicated to the models and algorithms used. As stated in the examples, the type of algorithm varies and basically four different approaches were applied in the scenario. This produces varied results and therefore improves the classification performance. If only the same algorithm is used, it is very likely that the results are very similar. Nevertheless, a scenario with differently trained or configured models of the same algorithm might also produce favorable results, in case the predictions vary sufficiently.
It has to be added, that a set of very efficient classifiers may not produce satisfactory improve or any improvement at all, since the predictions might already be close to the 100%. Anyhow, the vote-classification can be adjusted in many ways, and the performance is strongly dependent on the applied voting technique, feature space, algorithms, model configuration and performance. The following list summarizes the constraints, which should be adhered for a voting scenario. However, it does not entirely guarantee a favorable output.

1. A test set for calibration is required and classification performances of each classifier are not allowed to vary in a large range
2. Unequal algorithm performances or baldy trained classifiers may impact a vote scenario in an unfavorable way
3. A voting implementation should at least contain 3 models or more
4. Applied algorithms and models should use different approaches or parameters

4.1.10 Vote-Classification in a Distributed Environment

In a last scenario, the voting-classifier will be implemented in a distributed environment, as stated in [48]. The purpose if this is to check weather a voting scenario can be applied in a Smart Grid implementation. The following sections describe the changes done to the NSL-KDD dataset and further details on the developed scripts to simulate a distributed voting scenario.

Adaption of the KDD-NSL Dataset

As it can be found in [48, p. 10], every module has its own specialized training set. For HAN implementations, only DoS and Probing attacks were used for training. The NAN model used the best HAN Model and was also trained with U2R attacks. The WAN model implemented both the best HAN and NAN model, and was also trained with the whole attack range, including R2L attacks. To create training and test sets for this purpose, the function “makeDataset” in the library “KDDTools” was used, and suitable datasets were built. Therefore, the NSL-dataset with the complete attack-types was downloaded and every specific attack has been mapped to its matching intrusion pattern, according to the table in [18, p. 7]. Afterwards, the data was filtered for the specified intrusions. This process was carried out on each of the regular NSL-KDD training set as well as the testing set. As output, three different training and testing sets, only containing the suited attacks, were created. The additional difficulty value, which came with the downloaded dataset and included a 43rd feature, was not added to the data.

4.1.11 Creation of a Hierarchical IDS Communication

To built the hierarchical network infrastructure, each IDS module had several differently trained models assigned. For the HAN IDS, only two classifiers were used. It consisted of an AdaBoost and a SVM model, which were trained only for DoS and Probing intrusions. This way, the implementation was less resource intensive and predictions were carried out faster. This scenario was chosen to be comparably small, due to the problem that home-devices usually have to be cheap and do often not provide high computation power. Since voting does not really work with two classifiers, both models had to agree on the classification result. Otherwise, the data was passed to a higher instance. In the case of a HAN node, the next layer is the NAN node. These type of nodes were trained additionally with U2R attacks and used three classifiers. In the NAN-scenario, at least one additional classifier had to agree on the result of the best performing SVM model. Otherwise, the packet was again passed to the next level. WAN nodes had four attack classifiers and implemented the voting mechanism. The WAN Model contained basically the vote-classifier version which has been used before, as it is presented in Table 4.4.
To realize the whole scenario, the scripts “createDIDS_modules” and “DIDS_Vote” were created. The first script builds the specified datasets, saves the testing data and trains the models. In addition to that, the trained models are saved with the “makePersistent” function. The “DIDS_Vote” script loads the trained models as well as the data and executes the vote-classification. Since this is only an experimental version, a complex communication was not implemented. Instead, a simple control flow with if-conditions was created. This simulates the communication, in case a classification can not be done. The listing below shows this process.

```
for data_instance in testData:

Decision = 0

HAN_Vote = [HAN_AdaBoost.predict(data_instance) ,HAN_SVM.predict(data_instance)]

if HAN_Vote[0][0] != HAN_Vote[1][0]:

NAN_Vote = [NAN_AdaBoost.predict(data_instance),

NAN_DecisionTree.predict(data_instance),

NAN_SVM.predict(data_instance)]
```

```
if (NAN_Vote[2][0] != NAN_Vote[1][0]) or (NAN_Vote[2][0] != NAN_Vote[0][0]):
...
```

First, each data_instance of the test set is used to make a prediction. The predicted value of the HAN module is then stored in the HAN_Vote array. If the check of the condition fails, the NAN carries out the vote in the next step. The same procedure with storing predictions is applied here too, with the array NAN_Vote. Afterwards the if-condition checks, weather the SVM has at least one other agreeing classifier. Is this not the case, the regular vote scenario will be carried out with four models in the WAN-module.

Results

The tests were executed with the regular NSL sets, but with filtered attack types. For the HAN-Scenario, the HAN-specific test set, without U2R and R2L attacks, was used for training and classification. For the NAN, only the U2L instances were missing in the data. The voting scenario, as it is described in the last section, was carried out. The results were compared to the best performing classifiers of each node. The following results were achieved:

1. 2% Overall accuracy increase with the HAN classification
2. 1% Overall accuracy increase with the NAN classification
3. 3% Overall accuriacy increase with the WAN classification

These results show that even with smaller amounts of available models and an hierarchical detection mechanism, the vote-classification is still able to improve the performance. Nevertheless, a better accuracy resulted in a trade-off with the FP-rate. The FP rates increased as follows:

1. 3% FP increase with the HAN classification
2. 2.8% FP increase increase with the NAN classification

3. 4% FP increase increase with the WAN classification

This shows a quite unfavorable trade-off, if FP-rates should be low. At the same time, more discovered attacks are the consequence. More uncovered intrusions should always be preferred in a security sensitive network, as it is the case in smart grid applications. However, other approaches and implementations using votes might be able to lower FP-rates.

To demonstrate, how many instances are passed to the next classification system, a variable was set to count the packages which were passed on. This is used in order to check, weather the classifiers work and do not produce a high amount of communication.

1. in the HAN scenario, 912 of 9412 packets were passed to the NAN node, 642 of these were further passed to the WAN node
2. in the NAN scenario, 1593 of 9412 were passed to the WAN node

The transfer rate of roughly 10-20% of the predictions, proves an efficient usage of the communication system and results in a classification with an accuracy increase of 1-2%.

5

Conclusion

The aim of this work was to improve FP rates for an anomaly IDS in a Smart Grid scenario. The NSL-KDD Dataset was used to train and test a set of classifiers, which then were combined with a voting technique. Even though a Smart Grid dataset was not used and is not available at this point, the gathered information and conclusions can also be applied to Smart Grid scenarios. To achieve best possible results, different scaling methods were applied, and a formula for a vote weight calculation technique was developed. The code development and execution was carried out in a python environment. Classification functions of the Scikit-Learn library were used with the complementation of developed source code. In addition to the regular NSL-KDD testing scenarios, the dataset was also modified and a set of different voting techniques were carried out and compared. The reduction of FP-rates and with it, the improvement of classification accuracy is a complicated task. To this aspect comes the quite resourceful intensive application, since more than only one classifier has to execute a prediction. The additional cost of time and resources might not be applicable in every scenario. In contrast to this downside, several benefits have been discovered with voting. With the experiments carried out on the classifier-voting system, it has been observed that a voting technique was able to show, in several cases, significant increase in prediction accuracy. Nevertheless, a reduction of FP-rates in voting always reduced the overall possible correct predictions. When a combination of very efficient classifiers was used, the improvement of accuracy was either the same or only little significant. The positive aspect however, were dropping FP-rates, which is in many cases more favorable in machine-to-machine traffic, when the same accuracy can be achieved. On the long run, this can avoid much overhead for false prediction handling. Another aspect treated was the combination technique. Although the output is always dependent on the chosen classifiers and their performance, the weight balancing formula was able to produce favorable results in most of the test scenarios. This was a general approach and proved to be quite successful. In addition to the formula suggestion, other techniques were tested, with specific focus on reducing the FP-rate. It was found out, that is possible to reduce the false predictions with a primary model and an overruling-system, or by integrating the rate into the suggested formula. The primary model scenarios were specifically fitted to the model performances and this method can be an alternative, if the weight voting technique is not working. Also, a set of constraints was derived from the observations and documented in the thesis. Based on these findings, it might be possible to apply a successful voting mechanism in a Smart Grid network. Especially since there are not any attack scenarios available yet, clustering classifications or outlier detection might be applied in the future. Those algorithms often have lesser performance than supervised classifiers and a voting scenario might be able to improve the resulting accuracy. Also, due to the machine-to-machine generated traffic of Smart Grid applications, a feature space for efficient classification might be developed more easily than in Internet applications. This will result in strong classification algorithms which can be improved afterwards with voting. Even if the enhancements are limited to several percent, the output can avoid thousands of false detentions in the long run. Elaborated scenarios also showed the strong dependency on the chosen feature space, scaling method, individual classification performance and parameters, as well as the combination method, which can be applied in many different ways. In terms of FP-Rates, one scenario showed a decrease of false attack predictions by from 0.40% to 0.20%. Even though this does not seem much, it reduces the amount of FPs by 50%. For Smart Grid implementations, this can be crucial to avoid false alarms and stabilize the system behavior. In addition to that, defective packets, which might be produced due to system errors, can very likely be detected by an anomaly IDS. In general, the scenarios showed that a vote-application can produce favorable results in many cases and implementations may benefit from improved FP-rates for Smart Grid applications. By combining this technique with other elaborated anomaly detection improvements, this approach could find its way in reliable IDS system for the Smart Grid.

Bibliography

[1] al., Paul B. Kantor et: *Intelligence and security informatics.* IEEE International Conference on Intelligence and Security Informatics, 2005.

[2] Anderson, R. and Fuloria, S.: *On the security economics of electricity metering.* In *Smart Grid Communications, 2010 First IEEE International Conference*, 2010.

[3] Anderson, R. and Fuloria, S.: *Who controls the off switch?* In *Smart Grid Communications, 2010 First IEEE International Conference*, pages 96–101, 2010.

[4] Andres F. Murillo: *Review of anomalies detection schemes in smart grids.* Teleinformation and Automation Group, Rio de Janeiro, 2012.

[5] Animesh Patcha and Jung-Min Park: *An overview of anomaly detection techniques: Existing solutions and latest technological trends.* Computer Networks, pages 3448–3470, 2007, ISSN 1389-1286.

[6] Cabdud Wyeest: *Targeted attacks against the energy sector: Security response.* Symantec Security Research, 2014.

[7] Cheng-Ho Huang and Jhing-Fa Wang: *Multi-weighted majority voting algorithm on support vector machine and its application.* In *TENCON 2009 - 2009 IEEE Region 10 Conference*, pages 1–4, 2009.

[8] Chris A. Ciufo: *Modular choices simplify and future-proof m2m, wi-fi, and zigbee connectivity.* Digi-Key Electronics, 2012.

[9] Dietterich, ThomasG.: *Ensemble methods in machine learning.* In *Multiple Classifier Systems*, volume 1857 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin Heidelberg, 2000, ISBN 978-3-540-67704-8.

[10] Dr. Sri Niwas Singh: *Electric Power Generation: Transmission and Distribution.* PHI Learning, 2008, ISBN 9788120335608.

[11] Duda, Richard O., Hart, Peter E., and Stork, David G.: *Pattern Classification (2Nd Edition).* Wiley-Interscience, 2000, ISBN 0471056693.

[12] Durumeric et al.: *The matter of heartbleed.* In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 475–488, New York, NY, USA, 2014. ISBN 978-1-4503-3213-2.

[13] Eckert, Claudia: *Sicherheit im smart grid: Eckpunkte für ein energieinformationsnetz.* Alcatel-Lucent Stiftung fuer Kommunikationsforschung, 2011.

[14] European Parliament and Council: *Directive 2009/72/ec : concerning common rules for the internal market in electricity and repealing directive 2003/54/ec*, 2009. `https://www.energy-community.org/pls/portal/docs/1164180.PDF`.

[15] Farhangi, H.: *The path of the smart grid.* Power and Energy Magazine, IEEE, pages 18–28, 2010, ISSN 1540-7977.

[16] Gareth James: *Majority vote classifiers: theory and applications.* Dissertation, Stanford University, 2006.

[17] Giani et al.: *Smart grid data integrity attacks.* Smart Grid, IEEE Transactions on, pages 1244–1253, 2013, ISSN 1949-3053.

[18] H. Güneş Kayacık and A. Nur Zincir-heywood: *Using self-organizing maps to build an attack map for forensic analysis.* International Conference on Privacy, Security and Trust, 2006.

[19] Harper, Allen: *Gray hat hacking: The ethical hacker's handbook.* McGraw-Hill, New York, 3rd ed. edition, 2011, ISBN 0071742557.

[20] Hashmi, M., Hanninen, S., and Maki, K.: *Survey of smart grid concepts, architectures, and technological demonstrations worldwide.* In *Innovative Smart Grid Technologies (ISGT Latin America), 2011 IEEE PES Conference on*, pages 1–7, 2011.

[21] Institute for Energy and Transportation: *Smart grid projects outlook 2014*, April 2015. http://ses.jrc.ec.europa.eu/smart-grids-observatory, visited on 4/19/2015.

[22] Jan Sochman, Jiri Matas: *Adaboost classification - lecture script.* http://cmp.felk.cvut.cz/~sochmj1/adaboost_talk.pdf, visited on 6/2/2015.

[23] John McHugh: *Testing intrusion detection systems: A critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory.* ACM Transactions on Information and System Security, pages 262–294, 2000.

[24] John Steven, Gunnar Peterson, Deborah A. Frincke: *Smart-grid security issues.* IEEE Computer Society, 2010.

[25] Ju-Min Park and James Pearson: *In north korea, hackers are a handpicked, pampered elite*, 12/05/2014. http://www.reuters.com/article/2014/12/05/us-sony-cybersecurity-northkorea-idUSKCN0JJ08B20141205, visited on 4/20/2015.

[26] Leif E. Peterson, Scholarpedia: *K-nearest neighbor.* http://www.scholarpedia.org/article/K-nearest_neighbor, visited on 6/1/2015.

[27] Linda, O., Manic, M., and Vollmer, T.: *Improving cyber-security of smart grid systems via anomaly detection and linguistic domain knowledge.* In *Resilient Control Systems (ISRCS), 2012 5th International Symposium on*, pages 48–54, 2012.

[28] Liz Barris: *Legal,constitutional and human rights violations of smart grid and smart meters.* http://stopsmartgrid.org/wp-content/uploads/2013/10/Legal-Constitutional-and-Human-Rights-Violations-of-Smart-Grid-and-Smart-Meter pdf, visited on 4/7/2015.

[29] Lo, Chun Hao and Ansari, N.: *Consumer: A novel hybrid intrusion detection system for distribution networks in smart grid.* Emerging Topics in Computing, IEEE Transactions on, pages 33–44, 2013, ISSN 2168-6750.

[30] Lyle Ungar: *Machine learning - computer and information science, university of pennsylvanias.* Lecutre Wiki, 2014. `https://alliance.seas.upenn.edu/~cis520/wiki/index.php?n=lectures.boosting`, visited on 5/28/2015.

[31] Machine Learning Group at University of Waikato: *Weka - data mining software in java.* `http://www.cs.waikato.ac.nz/~ml/weka/`, visited on 6/4/2015.

[32] Mitchell, R. and Ing-Ray Chen: *Behavior-rule based intrusion detection systems for safety critical smart grid applications.* Smart Grid, IEEE Transactions on, pages 1254–1263, 2013, ISSN 1949-3053.

[33] Mitchell, Thomas M.: *Machine Learning.* McGraw-Hill, Inc, New York, NY, USA, 1st edition, 1997, ISBN 0070428077.

[34] Naruchitparames et al.: *Secure communications in the smart grid.* In *Consumer Communications and Networking Conference (CCNC), 2011 IEEE*, pages 1171–1175, 2011.

[35] NETL - National Energy Technology Laboratory: *The netl modern grid initiative,powering our 21st-century economy: smart grid benefits*, 2007. `https://www.netl.doe.gov/File%20Library/research/energy%20efficiency/smart%20grid/whitepapers/Modern-Grid-Benefits_Final_v1_0.pdf`, visited on 5/26/2015.

[36] Network Security Laboratory, University of New Brunswick: *Nsl-kdd data set for network-based intrusion detection systems*, 2009. `http://nsl.cs.unb.ca/NSL-KDD/`, visited on 5/26/2015.

[37] OpenCV: *Introduction to support vector machines.* `http://docs.opencv.org/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html`, visited on 5/27/2015.

[38] Pedregosa et al.: *Scikit-learn: Machine learning in python.* Journal of Machine Learning Research, pages 2825–2830, 2011.

[39] Remya, K. R. and Ramya, J. S.: *Using weighted majority voting classifier combination for relation classification in biomedical texts.* In *Control, Instrumentation,*

Communication and Computational Technologies (ICCICCT), 2014 International Conference on, pages 1205–1209, 2014.

[40] Shigeo Abe: *Support Vector Machines for Pattern Classification (Advances in Computer Vision and Pattern Recognition).* Springer, 2010, ISBN 978-1-84996-098-4.

[41] SIGKDD: *Kdd cup 1999: Computer network intrusion detection*, 1999. `http://www.sigkdd.org/kdd-cup-1999-computer-network-intrusion-detection`, visited on 5/19/2015.

[42] Tabrizi, F. M. and Pattabiraman, K.: *A model-based intrusion detection system for smart meters.* In *High-Assurance Systems Engineering (HASE), 2014 IEEE 15th International Symposium on*, pages 17–24, 2014.

[43] Tavallaee, M., Bagheri, E., Lu, Wei, and Ghorbani, A. A.: *A detailed analysis of the kdd cup 99 data set.* In *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, pages 1–6, 2009.

[44] Theuns Verwoerd and Ray Hunt: *Intrusion detection techniques and approaches.* Computer Communications, 2002. `http://www.researchgate.net/profile/Ray_Hunt/publication/222551704_Intrusion_detection_techniques_and_approaches/links/00b7d5266f587d2645000000.pdf`, visited on 6/2/2015.

[45] Tsang, R.: *Cyberthreats, vulnerabilities and attacks on scada networks.* `http://de.scribd.com/doc/144823067/Tsang-SCADA-Attacks#scribd`, visited on 5/26/2015.

[46] United States Department of Energy: *2014 2014 smart grid system report*, 2014. `https://www.smartgrid.gov/sites/default/files/doc/files/2014-Smart-Grid-System-Report.pdf`, visited on 4/19/2015.

[47] WhatIsSmartGrid.org: *Consumer benefits.* `http://www.whatissmartgrid.org/smart-grid-101/consumer-benefits`, visited on 6/2/2015.

[48] Zhang, Yichi, Wang, Lingfeng, Sun, Weiqing, Green, R. C., and Alam, M.: *Distributed intrusion detection system in a multi-layer network architecture of smart grids.* Smart Grid, IEEE Transactions on, pages 796–808, 2011, ISSN 1949-3053.

[49] Zhong Fan et al.: *Smart grid communications: Overview of research challenges, solutions, and standardization activities.* Communications Surveys Tutorials, IEEE, pages 21–38, 2013, ISSN 1553-877X.

List of Abbreviations

WAN Wide Area Network

NAN Neighborhood Area Network

HAN Home Area Network

IDS Intrusion Detection System

PLC Power Line Connection

SGDIDS Smart Grid Distributed Intrusion Detection System

FP False Positive

FN False Negative

TP True Positive

TN True Negative

U2R User to Root

R2L Remote to User Local

DoS Denial of Service

SVM Support Vector Machine

RBF Radial Basis Function

kNN K-Nearest Neighbor