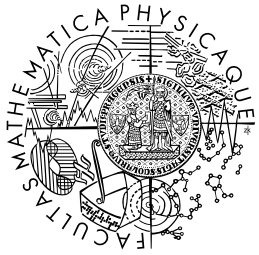


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Viktor Vašátko

Applications of Artificial Intelligence in IT Security

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis: Mgr. Marta Vomlelová, Ph.D.
Study programme: Computer Science
Study branch: Artificial Intelligence

Prague 2020

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to thank my supervisor, Mgr. Marta Vomlelová, Ph.D., for her constructive comments, advice and guidance. I also would like to thank my family for their support and patient during my studies.

Title: Applications of Artificial Intelligence in IT Security
Author: Viktor Vašátko
Department: Department of Theoretical Computer Science and Mathematical Logic
Supervisor: Mgr. Marta Vomlelová, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: The objective of this work is to explore the intrusion detection problem and create simple rules for detecting specific intrusions. The intrusions are explored in the realistic CSE-CIC-IDS2018 dataset. First, the dataset is analyzed by computing appropriate statistics and visualizing the data. In the data visualization various dimensionality reduction methods are tested. After analyzing the dataset the data are normalized and prepared for the training. The training process focuses on feature selection and finding the best model for the intrusion detection problem. The feature selection is also used for creating rules. The rules are extracted from an ensemble of Decision Trees. At the end of this work, the rules are compared to the best model. The experiments demonstrate that the simple rules are able to achieve similar results as the best model and can be used in a rule-based intrusion detection system or be deployed as a simple model.

Keywords: machine learning security rules

Název práce: Aplikace umělé inteligence v IT bezpečnosti
Autor: Viktor Vašátko
Katedra: Katedra teoretické informatiky a matematické logiky
Vedoucí diplomové práce: Mgr. Marta Vomlelová, Ph.D., Katedra teoretické informatiky a matematické logiky

Abstrakt: Cílem této práce je prozkoumat problematiku detekce útoků na počítačové systémy a vytvořit jednoduchá pravidla, která jsou schopna detekovat jednotlivé útoky. Útoky jsou prozkoumány na realistickém datasetu CSE-CIC-IDS2018. Nejprve se práce zabývá analýzou datasetu. V analýze jsou spočítány různé statistiky datasetu a na závěr jsou otestované různé metody redukce dimenzí pro zobrazení dat v dvou demenzionálním prostoru. Po analýze následuje příprava a normalizace dat. Proces trénování se pak zaměřuje na výběr vhodných příznaků a hledání nejlepšího modelu. Stejné příznaky jsou pak použity i pro vytváření pravidel. Pravidla jsou extrahována ze souboru rozhodovacích stromů. V závěru práce jsou pravidla porovnána s nejlepším modelem. Experimenty ukazují, že jednoduchá pravidla jsou schopna dosáhnout podobných výsledků jako nejlepší model. Mohou být použita v pravidlových systémech pro detekci útoků nebo nasazena jako jednoduchý model.

Klíčová slova: strojové učení bezpečnost pravidla

Contents

Introduction

Many companies and people are interested in IT security. Since the first bigger computer systems and computer networks were created, IT security became important, but these days it is much more important than before. Many services are dependent on some computer systems and without these computer systems they are not able to work. We all are tied together with computers that we even can not imagine losing our phones or the Internet and services on it, but this is not everything. Everyone has some private data somewhere in data storage, which may be, but it does not have to be, connected to the Internet. These data are also very important for us and we do not want to provide them to someone unauthorized. There are people that want to get access to the private data or want to cause a damage to some services. That is why everyone wants to protect their own services and their own data. This is the reason why some researchers are interested in IT security and try to improve it.

However, IT security is a general term and we can imagine many things, which can be called IT security. In one point of view, it can be a building of secured networks, computer systems or even secured warehouses with a data storage. In another point of view, it can be some mechanism of intrusion detection or anomaly detection. An intrusion detection system (IDS) can monitor our environment and detect if something strange is happening. It can provide a possibility to defend ourselves. The biggest problem is that it is impossible to create a 100% secured area. It is always trade off between security and price for security as one well-known graph explains (see Figure 1). It basically says the more resources are invested, the more secured area is, but as Figure 1 demonstrates, an area never will be secured for 100%. People will always have to deal with a question what level of security is enough for us. However, in some level of security, an intrusion is not worth it or is very expensive.

Intrusion detection will be always needed at least until someone will create a 100% secured computer system. This work focuses on intrusion detection. Intrusion detection is a very popular subject in IT security. Attackers create new intrusions and new methods, so researchers in IT security have to create new defense and adapt intrusion detection systems. With the rise of machine learning (ML) many researchers tried to apply it on this problem, but ML needs data to learn to detect an intrusion. Creating a good and realistic dataset for intrusion detection is not easy. Many datasets were proposed, but some of them contains anonymous data or are not very realistic. The Communications Security Establishment and the Canadian Institute for Cybersecurity dealt with creating suitable datasets and the CSE-CIC-IDS2018[1][2] dataset was created. This dataset is used for creating ML models.

ML models for intrusion detection are mapped very well, especially ML models that are trained on the CSE-CIC-IDS2018 dataset. However, not every ML model can be easily interpreted. A ML model is usually black box, which takes an input and gives us a prediction, but what if a human expert wants to know how it decided and wants to understand the intrusion. One solution may be to use a simple and understandable ML model such as Decision Tree. In Decision Tree, it is easier to track its decisions, but the model may be still too complex. It brings

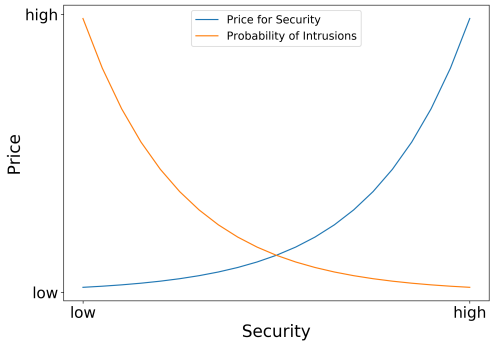


Figure 1: Trade off between security and price.

us to the second solution – creating a set of simple and short rules. Rules are logical formulas, which can define an intrusion and are very easy to understand. However, they may be less powerful than other ML models. The objective of this work is to create simple and short rules for each type of the intrusion in the dataset and to achieve similar results as the best ML model achieves.

Related Work

The CSE-CIC-IDS2018[1][2] dataset is not the first dataset created by the Communications Security Establishment and the Canadian Institute for Cybersecurity. They created another similar intrusion detection datasets. One of them is called CICIDS2017[3][4]. Both datasets are created in the same way. However, the CICIDS2017 dataset simulates only five days of attacking and normal activity and the simulated network is small in comparison with the CSE-CIC-IDS2018 dataset. The CSE-CIC-IDS2018 dataset simulates ten days of attacking and normal activity. Its simulated network consists of several subnets to get a network similar to real-world networks. In addition to the CICIDS2017 dataset, Iman Sharafaldin et al.[5] propose several ML models. They mainly focus on analyzing the dataset. Before training the ML models, they study feature importance of each intrusion in the dataset and suggest which features are the most important. For extracting feature importance, they use Random Forest. Random Forest is also used in the final testing of the most important features together with K-Nearest Neighbors, ID3, Adaboost, Multilayer perceptron, Naive-Bayes and Quadratic Discriminant Analysis. Their results show that Random Forest is the best ML model for their dataset. At the end of their work, they compare available datasets and show in what their dataset differs.

The CSE-CIC-IDS2018 dataset is studied by Qianru Zhou and Dimitrios Pezaros[6]. In their work, they use the same ML models as in [5]. ML models are trained on the CSE-CIC-IDS2018 dataset, but for testing, they use their real-life traffic data. The results show that the best classifier is Decision Tree, but also other ML models perform well.

V. Kanimozhi and T. P. Jacob[7] propose Multilayer Perceptron. They focus only on Botnet intrusion and hyper-parameter optimization. They achieve similar results, as in [6].

There is also a different approach. T. Chadza et al. propose Hidden Markov Model[8]. They study the detection accuracy of determining all states, the current state, and the prediction of the next state of an observation sequence. For training they use two algorithms Baum Welch and Viterbi Training. For initialization of the Hidden Markov Model, they use three methods – random, uniform and count-based. At the end of their work, they present a comparison of initialization methods and their influence on accuracy.

Before the datasets created by the Communications Security Establishment and the Canadian Institute for Cybersecurity, there were one more widely used dataset for the evaluation an IDS. The dataset is called KDD Cup 1999[9] or also only KDD'99. It consists of 4 898 431 network traffics and each traffic has 41 features. The network traffics are divided into 5 categorizes. Each category represents a group of attacks with similar characteristics. One of the category is normal activity. In total there are 22 attacks. It is old dataset and it was sometimes criticized that the data are too "artificial".

The fact that the dataset is too "artificial" is partially discussed by Ugo Fiore et al.[10]. They realize that the environment, where IDS has to work, is very variable. The attacks differ, new attacks are coming and each network is different. The main goal was to provide a more adequate description of network

traffic and to be as adaptive as possible. For this reason, they propose semi-supervised anomaly detection with using Discriminative Restricted Boltzmann Machine (DRBM) that is an energy-based, generative model not restricted to any specific environment, or a priori knowledge base. The training, in semi-supervised anomaly detection, contains only samples representing normal activity. Then what can not be characterized as normal activity is marked as anomalous. They did two experiments. The first experiment was to train and test the DRBM on the real data. The second experiment was to train the DRBM on the KDD'99 dataset and to test it on the real data. At the end of their work, they compare the results from the both experiments and show that the performance suffers if the classifier is tested in the different network than where it was trained.

The Restricted Boltzmann Machine (RBM) is also used in another form. The RBMs are stacked to create Deep Belief Neural Network (DBN). The DBN is a generative neural network. Mostafa A. Salama et al.[11] use the DBN as a dimensionality reduction method together with SVM classifier. The whole model is trained on the NSL-KDD dataset. It is only the improved KDD'99 dateset. The goal is to classify 5 different groups of attacks. First, they study how well the DBN works as a dimensionality reduction method. It is compared to PCA, Gain Ratio and chi square. For each case, the SVM is used as a classifier. The results demonstrate that the best dimensionality reduction method is the DBN. Then they compare three classifiers – SVM, DBN and combination of DBN and SVM. The models are trained on three different sizes of the training set. Only 20%, 30% and 40% of the dataset is used for the training. The experiments show that the performance of models is improving with increasing size of the training set. At the end of their work, the combination of DBN and SVM is proposed as the best model. However, the DBN is also used by Md. Zahangir Alom et al.[12]. In their work, the DBN is used only as a classifier. It is trained on the same dataset with the same sizes of the training set. Their DBN performs better than the models in [11].

Jihyun Kim et al.[13] use different approach. To take in account long term dependencies, they use Long Short Term Memory (LSTM). The LSTM is a recurrent neural network (RNN) that improves the problem with a vanishing and exploding gradient in the back propagation training of a RNN. The LSTM is trained on the KDD'99 dataset but on the version of the dataset with only 10% of data. It is trained to detect 5 different groups of attacks. First, they search for optimal hyper-parameters and then they test the trained LSTM. The LSTM achieves very good results. Wafaa Anani and Jagath Samarabandu[14] study efficiency of the different types of LSTM architecture. The architectures are – LSTM, Gated Recurrent Unit (GRU), Bidirectional LSTM (Bi-LSTM) and Skip-RNN. To train the models, they use the KDD'99 10% dataset. The training does not use all features but only twelve the most important features. Importance of the features is computed by Random Forest that is very effective on the KDD'99 dataset. At the end of their work, they present results for 100, 500 and 1000 training cycles. The LSTM outperforms all architectures. Only for 1000 training cycles, the LSTM is outperformed by the GRU. The LSTM performance is better than performance of previous mentioned models for the KDD'99 dataset.

Outline

This work is structured as follows. Chapter 1 summarizes information about the dataset – the dataset origin, features and goal classes. The chapter also includes intrusions overview to have some idea about various ways of attacking. Chapter 2 reviews the Machine Learning theory. It describes the basic concepts of Machine Learning and its goals. It also discusses various types of ML models, which are used in experiments and for comparison with rules. The ML models discussion is followed by reviewing the method for creating rules. The chapter ends with a short discussion about unsupervised learning. Chapter 3 presents the model training. It goes through the process of data preprocessing, data visualization, feature selection and ends with the training of several models. The last chapter is Chapter 4. It discusses rules creation and presents the final rules. The rules are created for each intrusion and use only the most important features. In addition to the rules creation, the rules are compared to the best ML model.

1. Intrusion Detection Dataset

This chapter covers the basic properties of the CSE-CIC-IDS2018[1][2] dataset. Section 1.1 summarizes how the dataset is created and how the intrusions are simulated. In Section 1.2, it continues with the data description. It describes the content of the dataset, its features and classes. The chapter ends with Section 1.3 focusing on the intrusions overview. Let us now look at the dataset closer.

As mentioned before, there are many datasets. The rules are inferred from the CSE-CIC-IDS2018 dataset. The CSE-CIC-IDS2018 dataset is one of the datasets for intrusion detection systems (IDS). It is created by the Communications Security Establishment (CSE) and the Canadian Institute for Cybersecurity (CIC). They focus on creating a realistic dataset with not anonymous data that most of such datasets contain. They also propose a method of dynamic dataset creation. It means not to create only one static dataset, but to be able to easily create a dataset in anytime and for any network topology. The CSE-CIC-IDS2018 dataset is not their first dataset. They have two more datasets related to IDS. First one is the CICIDS2017[3] dataset and the second one is the ISCXIDS2012[15] dataset. The datasets are created using a similar method as in the CSE-CIC-IDS2018 dataset, but the dataset from 2018 uses different network topology and the network is simulated on AWS[2] (Amazon Web Services).

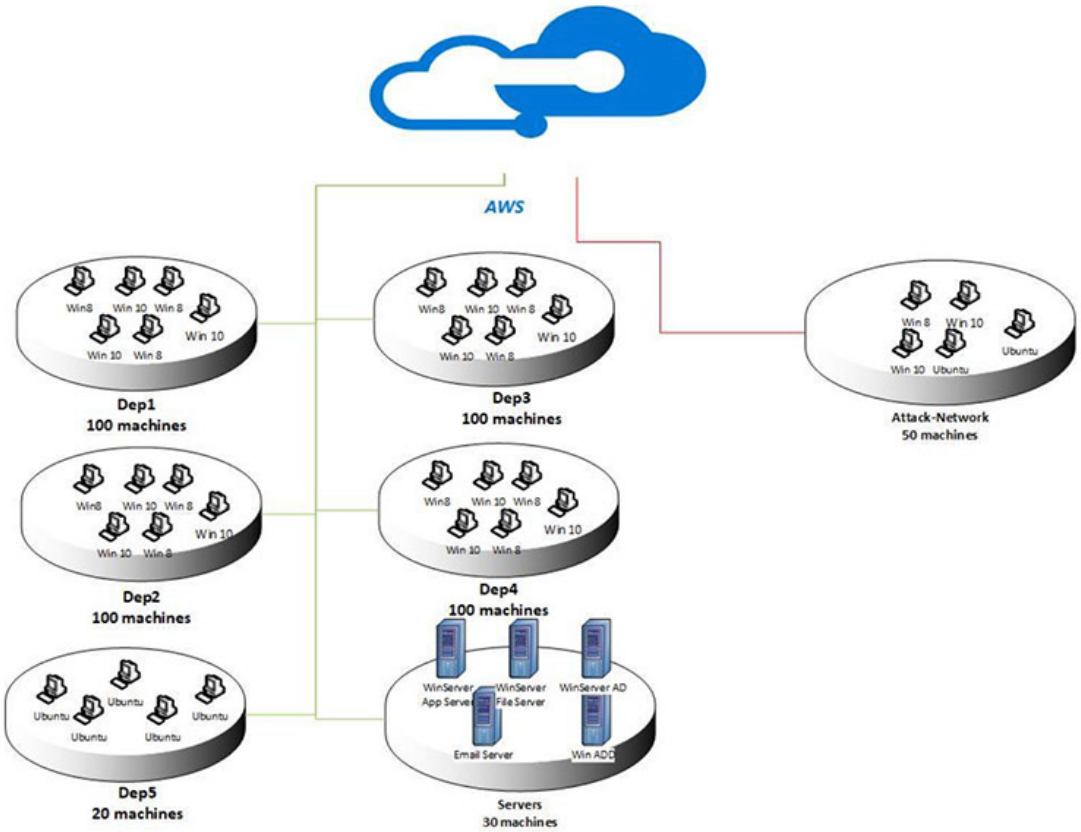


Figure 1.1: Network topology[1].

1.1 Dataset Creation

CSE and CIC did not only create the CSE-CIC-IDS2018 dataset but also came up with a method how to easily generate datasets for specific needs with different network protocols and topology. The main idea is to extract characteristics from a normal activity and from an activity when someone tries to attack a network. Based on these characteristics, called profiles, agents, or human operators are able to generate specific events in a network. Using various machine learning methods and statistical analysis techniques, they put together two profiles. The first B-profile is profile describing the activity of usual users. With this profile it is possible to generate benign flow. The second M-profile is profile describing an attack scenario in an unambiguous manner. They considered six different attack scenario – DoS, Web Attack, Infiltration, Botnet, DDoS and Brute Force Attack.

First of all, they prepared a test environment to generate the final dataset. The test environment was a common LAN network on the AWS computing platform. To be close to some real network as much as possible, they divided the LAN network into five subnets and used 20-100 machines in each subnet. Each subnet contained various machines with various operating systems. Attacker's network was outside of the LAN network. Attacker's network contained fifty machines to be able to attack from one or more machines. The whole network topology can be seen in Figure 1.1. Then they simulated a normal activity and intrusions for ten days. Each day was recorded. They recorded the whole flow. However, raw data are not suitable for ML models. The data has to be prepared for ML models first. They used their feature extractor CICFlowMeter-V3[16] and extracted 80 features for ML models.

1.2 Description

The dataset is divided into ten days. For each day they provide separate files. In the files, we can find specific intrusions and a normal activity called benign. To provide a possibility to create a new feature extractor or use different feature extractor the dataset consists of two types of files. The first type contains only raw data. Raw data contain both network packets and event logs from machines. Network packets are stored in pcap files used for analyzing a network. It is not possible to use directly raw data for machine learning, that is why they also provide the second type of files – CSV. CSV files contain 80 extracted features together with labels (see Table A.2). The list of files and what intrusions they contain is attached in Attachment A.1. The features contain basic information such as Protocol, Destination Port, Timestamp and then many statistics describing data flow. Some features contain invalid values, specifically infinity (*inf*) and not a number (*nan*). Also some features have only one distinct value.

The dataset contains common and well-known intrusions. It has 15 classes. One of them is Benign representing a normal activity and rest of them are various types of intrusions. The main types of intrusions are Brute Force, Denial of Service (DoS), Distributed Denial of Service (DDoS), SQL Injection, Infiltration, Web Attacks and Botnet. The main types are then divided into classes based on which software is used to make the intrusion. Some of the classes do not have very large occurrence. The dataset has 16 232 943 samples and size approximately 7

GB. Below you can see the list of classes, their occurrence and in Figure 1.2 their ratio.

- Benign - 13 484 708 (83.0700139%)
- Bot - 286 191 (1.7630260%)
- Infiltration - 161 934 (0.9975640%)
- SQL Injection - 87 (0.0005359%)
- Brute Force -Web - 611 (0.0037640%)
- Brute Force -XSS - 230 (0.0014169%)
- FTP Brute Force - 193 360 (1.1911580%)
- SSH Brute Force - 187 589 (1.1556068%)
- DDoS attack-HOIC - 686 012 (4.2260482%)
- DDoS attack-LOIC-UDP - 1 730 (0.0106573%)
- DDoS attacks-LOIC-HTTP - 576 191 (3.5495166%)
- DoS attacks-Hulk - 461 912 (2.8455222%)
- DoS attacks-SlowHTTPTest - 139 890 (0.8617661%)
- DoS attacks-Slowloris - 10 990 (0.0677018%)
- DoS attacks-GoldenEye - 41 508 (0.2557022%)

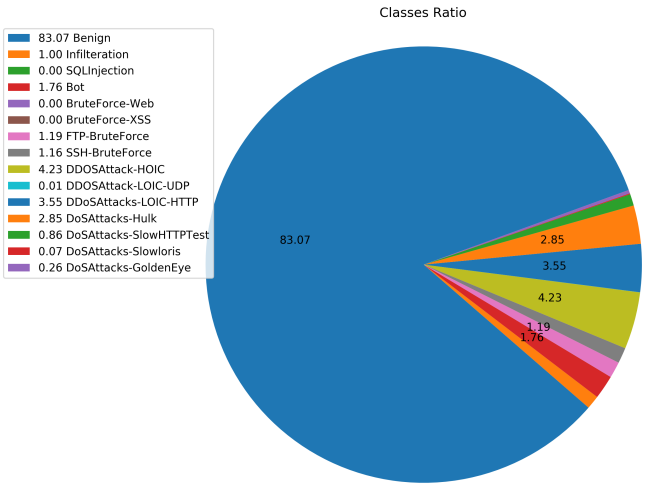


Figure 1.2: Classes ratio.

1.3 Intrusions

This section discusses how specific intrusions differs from each other and how intrusions look like. With knowledge about intrusions, it may be easier to look for an anomaly in the data. Below are only common intrusions contained in the dataset, but in the real world can be found many other intrusions. Attackers permanently adapt to new defense and develop new methods of how to get to their target.

Botnet

The word botnet[17] comes from two words robot and network. Attacker's objective is to infect computers in the targeted network with malicious software and join them into a network of robots. After that, they are able to control these computers and can make various types of attack. With many controlled computers, it is easy to make DDoS attack, but also it is possible to steal data or sell an access to them. If someone becomes a part of botnet attack, they may be legally responsible for the consequences[18].

For simulating botnet, attack they used two software in the dataset. The first software is Zeus. Zeus is a Trojan horse malware used mainly for stealing data like banking information or keystroke. As a complement, they used second software Ares botnet which is capable of running remote commands, download/upload and other useful actions. After infecting computers, they requested screenshots from them every 400 seconds and nothing more.

Brute Force Attack

The whole idea of a brute force attack is trial-and-error[19]. This type of attack is used for a password and encryption key cracking. There is nothing clever on this method. As the name of the attack suggests, it is only a brute force method. It may take a lot of time to complete the attack successfully. On the other hand, for some situations, it is enough and it works very well. A simple example is a password cracking when every combination is tried to find the correct password.

There are many tools for brute force attack. They chose Patator, which is written in Python, to implement a brute force attack in this dataset. They implemented two types of attack FTP and SSH. Attacks were coming from Kali Linux system and were aimed at Ubuntu 14.0 system. They also had a large dictionary with 90 million words for a password cracking.

Denial of Service (DoS)

Each service or computer system has some limited capacity to handle requests from users. If service or computer system receives too many requests, it becomes slower or it can even crash. Denial of Service attack (DoS) aims to get a target to this situation. The attack comes from one machine and tries overwhelming or flooding a target. DoS attack has two typical categories[20] – buffer overflow attack and flood attack. Buffer overflow attack tries to consume all available resources such as hard disk space, memory or CPU time. In flood attack, the

objective is to create a big amount of packets sent to a server, which cause server oversaturating.

In this scenario, they used the Slowloris Perl-based tool to take down a web server with the minimal bandwidth and side effects. The attack was led by a single machine.

Distributed Denial of Service (DDoS)

As can be deduced from the name of the attack it is a similar attack like DoS. The only difference is that the attack is led from multiple machines. For this purpose, botnet attack can be used to take a control over multiple machines and organize them to attack some server. In the dataset, they used two tools – High Orbit Ion Cannon (HOIC) and Low Orbit Ion Cannon (LOIC). For example, the attack with HOIC was led from 4 computers.

Infiltration

Infiltration is the type of attack which tries to do a damage or map a network from inside. Typically this is done by sending some malicious software to someone in a network and creating a backdoor. Then the attacker can access a network from the backdoor and scan its topology. To implement this type of attack, they used a malicious document. The document was sent by email and then they were able to map a network.

Web Attacks

Web attacks are a group of attacks targeting on web pages. Into this group can be included, for example, cross-site scripting[21] (XSS) or SQL injection[22]. XSS attack aims at the user's web browser. It tries to run some malicious script on the user's side. The user may find these scripts during usual browsing the Internet. A malicious script might be injected using a web page's input. If some web page does not check an input and use that to generate an output, it may be in a risk. Common targets are web pages with forums, comments or message boards. Compared to SQL Injection, SQL Injection aims at databases behind a web page. It tries to go around an authentication and authorization and manipulate with data using SQL statements. Without a good protection, the attacker may add, delete or modify records in the database or even retrieve some sensitive data. SQL Injection is one of the oldest attack.

To simulate this type of attack, they used web application Damn Vulnerable Web App (DVWA), which is vulnerable and used for testing, teaching and helping to better understand the process of securing a web application. They implemented SQL Injection attack, XSS attack and some brute force attacks. For automation attacks, they developed a code with Selenium framework.

2. Machine Learning

This chapter reviews the Machine Learning theory. The Machine Learning theory and its goals are introduced in Section 2.1. Section 2.2 discusses the supervised learning. It goes through the training process, loss functions and regularization, metrics and ends with presenting several Machine Learning models and the method for creating rules. Section 2.3 discusses the unsupervised learning. It focuses mainly on methods for the dimensionality reduction, that are helpful in the data visualization. Let us start with an introduction.

2.1 Introduction

Machine Learning (ML) and Deep Learning (DL) are popular and quickly growing fields of research. People are trying to apply ML and DL on various kinds of problems. A current very popular field of research is, for example, natural language processing (NLP), where researchers try to create models, that are able to understand natural languages such as English language or Czech language. In this field of research, many problems can be found, such as language translation, sentiment analysis or smart assistants. Another popular field of research is Automated Machine Learning (AutoML), which tries to automate the process of ML and make ML accessible for everyone.

Both ML and DL study how the computer can learn to do some task. The task can be, for example, NLP, AutoML or something less complex like simple logical function *AND* or *OR*. This work focuses on ML. DL is more associated with large and complex artificial neural networks, but they are not necessary here. There are many definitions of ML, that have various point of views, but let us mention only one definition. American computer scientist Tom M. Mitchell defines the computer learning as: *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."*[23]. Task T is already clear, but others are not. Let task T be intrusion detection then experience E is correctly and incorrectly detected intrusions and performance measure P can be how many intrusions are correctly detected. ML needs some examples to get experience E.

In this work, the main objective is to use ML to detect intrusions and explore their characteristics. ML needs data to learn something. Let $X \in R^{n \times m}$ be data with n rows and m columns. Rows are called samples. Each sample is m dimensional row vector $x_i \in R^m$. The elements of the vector $x_i = (x_{i1}, x_{i2}, ..., x_{im})$ are called features or attributes. In the case of intrusion detection features are the destination port, protocol and all statistics about a flow.

In some cases of ML, only data X are provided. Data X do not have any information about what each sample means. Based on what is provided, four types of ML are differentiated – supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning. This work uses only supervised learning and unsupervised learning.

Supervised learning also called learning with a teacher, is a method, which uses data X and their labels. Labels are defined as a column vector $Y \in R^n$ and

$Y^T = (y_1, y_2, ..., y_n)$. It tries to learn to predict for a sample x_i a value y_i. If Y is a real vector, it is called regression. Examples of regression tasks are predicting stock price, house price or the age of person. If Y is a discrete vector, it means $Y \in \zeta^n$, where ζ is a finite set of possible outcomes, then it is called classification. Intrusion detection is an example of classification. In intrusion detection the set ζ can have two forms. In case of predicting only if it is an intrusion or not, ζ has only two values $\{0, 1\}$. This type of classification is called a binary classification. Predicting 0 means it is not an intrusion and predicting 1 means the opposite. However, an intrusion detection system can have requirements also to detect what type of an intrusion it is. In this case, ζ can have more than two values. It is called multiclass or multinomial classification.

Not every data X are labeled. Someone has to label the data. It requires many people for big data and for specific data, it requires people with knowledge about the problem. ML has to deal with unlabeled data too. Even without labels, ML should be able to extract some information from the data. Learning without labels is called unsupervised learning. In this case, ML has no information about what it is. In unsupervised learning, ML tries to find relations, similarities and guess possible classes, which might or might not correspond to the reality. In intrusion detection, it can be applied to discovering how many different behaviors are in the data. Then some human expert can say that some behaviors are usual and some are not usual. Based on cooperation ML and the human expert, the data can be easily split into negative and positive samples without seeing each sample. Other unsupervised learning methods help with the data visualization and dimensionality reduction. Graphs can represent a maximum of 3-dimensional space, but samples may have tens, hundreds or even thousands dimensions. These methods can reduce their dimensions and preserve some of their dependencies. Let us now go through supervised and unsupervised learning.

2.2 Supervised Learning

The general concept of supervised learning is to find a hypothesis h. Let χ be a set of all possible samples of a machine learning problem and ζ be a set of all possible outcomes for the classification problem. Then the hypothesis h can be defined as a function $h : \chi \to R$ for regression and $h : \chi \to \zeta$ for classification[24]. The process of learning can be defined as searching for the most suitable hypothesis h in the space of hypotheses H. For finding such a hypothesis, it needs data and a performance measure. Let $X \subseteq \chi$ and $|X| = n$ be some data and $Y \in R^n$ or $Y \in \zeta^n$ be their labels, then the performance measure can be defined as function $f : Y, \hat{Y} \to R$, where $\hat{Y}$ are hypothesis's predictions $h(X) = \hat{Y}$. The measure defines what the hypothesis should learn, the objective. Performance measures that measure the error of hypothesis are called loss functions or also error functions. These functions are important in optimizing and searching for the most suitable hypothesis. However, these functions are not suitable for measuring how much the hypothesis is accurate. For this purpose, there are various accuracy functions or also called metrics. Some of them are similar to loss functions, but some of them give different results.

ML has many different models and each of them has different capabilities. Some of them are more complex and able to learn more difficult tasks and the

others are less complex, but may be faster and interpretable. It does not mean that the most complex model is the best model. For some tasks, it is more preferable to understand why some decisions were made, but this is not the only reason. A set χ may contain some noise. It can be noise from some sensors, or it can be mistakes during a measurement. A more complex model is capable of learning more things than a simpler model, so more complex model may learn even the noise and then provide incorrect predictions. This phenomenon is referred to overfitting. On the other hand, a simpler model does not have to be able to learn difficult problems. This is called underfitting. There is always a question how complex model to choose, but there is one rule defined by Occam's razor. Here the paraphrased Occam's razor is: *"The simplest solution is most likely the right one."*. The simplest model capable of learning the problem is most likely the right one.

2.2.1 Training Process

The whole process of choosing and training a model starts with a data preprocessing. Usually data come in a raw form not suitable for training a model. The data preprocessing contains several phases. First of all, if data are in a raw form, features have to be extracted. For example, in the CSE-CIC-IDS2018[1] dataset features are extracted using CICFlowMeter-V3[16]. Most of the features are traffic statistics. Features are usually some statistics or attributes such as categorical (Protocol, Color, etc.) or numerical (Flow duration, Size, etc.). When the features are extracted, the feature engineering can be used. Some problems have not enough features and then it is difficult to find a good model. The feature engineering is about creating new features. A new feature can be created by combining existing features with using various operations such as multiplication, division, or even some ML model can provide new features. Sometimes data are not complete. Some features may contain missing values or invalid values. These values are not suitable for models and models are not able to handle that. If data are large, it is possible to drop incorrect samples, but if data are small, it is not possible. For example, missing values can be filled in with the most common value of the feature, but it always depends on the feature. Now data are almost prepared, but one more modification can be done. Some models prefer normalized data than data with big intervals of numbers, that is why data are scaled into specific small intervals. Usually standardization by standard deviation and mean or min-max scaling are used.

In the final step of data preprocessing, the data are split to a training set and test set. The training set is used for a model training and the test set is used for a model evaluation. The test set must not be used for the training, because it would learn correct answers and the evaluation would not provide a valid result of how well model is able to predict unseen data. Most of the models have many hyperparameters that can affect the result of training. Tuning these hyperparameters is done on a validation set. The validation set is taken from the training set and the rest of the training set is used for the model training with specific hyperparameters. After finding the best hyperparameters the model is trained on the whole training set and tested on the test set. These basic steps can be followed in most of the ML problems.

2.2.2 Loss Functions and Regularization

There are many loss functions and each of them is suitable for different ML tasks and different models. The loss function is very important because it defines the objective of what the model should learn. The model training is basically optimizing one of the loss functions. The loss functions measure how many and how big mistakes the model makes. The model that does not make any mistakes in the training has the loss function equal to 0. The optimization problem is the minimization of the loss function. However, some loss functions together with some models may be very complex and a finding of global minimum may be very difficult. Usually, the optimization problem ends in some appropriate local minimum. For each ML task, there are different loss functions. Regression has its own specific loss functions and also classification has its own loss functions. Let us define only the classification loss functions.

The classification loss functions work with classes and probability distributions. This is the reason why it is not appropriate to use a regression loss function for a classification problem. One of the loss functions is Cross-Entropy loss. The Cross-Entropy loss is defined as:

$$L = -\sum_{i=0}^{n}\sum_{c=0}^{m} y_{i,c} log(\hat{y}_{i,c})$$

where $y_{i,c}$ is the indicator if *i-th* sample belongs to class c and $\hat{y}_{i,c}$ is predicted probability that *i-th* sample belongs to class c. The binary case is defined as:

$$L = -\sum_{i=0}^{n} y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)$$

The advantage of this loss function is that it much more penalizes the predictions, that are far from the true class, and less penalizes the predictions that are close to the true class. If the model predicts 0.8 and the true class is 1, then it already says that the class is 1 and it does not need to be penalized a lot. In comparison with only subtracting the true class and predicted class (Mean Absolute Error) the Cross-Entropy gives error 0.097 and the subtracting gives 0.2.

Another loss function is Hinge loss. Hinge loss is related with Support Vector Machine and its maximum margin classification. The loss function is defined as:

$$L = \sum_{i=0}^{n} max(0, 1 - \hat{y}_i y_i)$$

where $y_i \in \{-1, 1\}$ is a true class and $\hat{y}_i \in R$ is a predicted value. There is no probability distribution. If the value is negative, then it is a negative class. If the value is positive, then it is a positive class. To get zero loss, the values $\hat{y}_i$ and y_i have to have the same sign and their product has to be greater than 1.

Next loss function is Exponential loss. Exponential loss is used, for example, in the AdaBoost algorithm. The loss function uses the exponential function:

$$L = \sum_{i=0}^{n} e^{-y_i \hat{y}_i}$$

If the values $\hat{y}_i$ and y_i have the same sign, then the loss is low. If the values have a different sign, then the loss grows very quickly. There are also other loss functions, but let us now move on to regularization.

The regularization is a technique on how to control the model complexity and avoid the overfitting. The ML models may have many various parameters. Sometimes it is appropriate to keep these parameters simple. For this purpose, there are methods that modify loss functions and add a penalization for the model complexity. One of the methods is L1 regularization. Let $\theta = \{w_1, w_2, ..., w_n\}$ be a set of model parameters and $w_i \in R$ be a parameter. Then the L1 regularization can be defined as:

$$L_\theta = L + \lambda \sum_{i=0}^{n} |w_i|$$

where L is one of the loss functions and λ is the weight of penalization for the model complexity. The L1 regularization generally keeps the model parameters close to 0. It is used, for example, in Logistic Regression. The L1 regularization together with Logistic Regression can be used for extracting a feature importance. In Logistic Regression, the number of parameters corresponds to the number of features. The L1 regularization shrinks the coefficients of less important features to 0. The features with a coefficient equal or close to zero are unimportant. The higher the coefficient is, the more important the feature is.

Another regularization technique is L2 regularization. Instead of using the absolute value, it uses square. The L2 regularization is defined as:

$$L_\theta = L + \lambda \sum_{i=0}^{n} w_i^2$$

This regularization technique is a little bit different. The parameters less than one does not give a big complexity, but the complexity grows very quickly with parameters greater than one. The L2 regularization does not help to shrink the parameters close to zero but helps to keep the parameters very small.

2.2.3 Metrics

After training a model, it needs to be decided how well the model performs on the task. There can be many different metrics. It always depends on what is important for the task. In Figure 2.1 is the Confusion Matrix, which describes how many values are correctly predicted and how many values are miss classified. There are four values TN, FN, TP and FP. The value TN means *True Negative*. A model predicts $0/False$ and the actual value is $0/False$. The value FN means *False Negative*. A model predicts $0/False$, but it is $1/True$. The value TP, called *True Positive*, is similar. A model predicts positive and it is positive. And the last value FP, called *False Positive*, means predicting positive, but the actual value is negative.

Each task may require a different performance. For example the intrusion detection task requires to detect all intrusions, because not detecting an intrusion may cause a damage to services or loosing private data. It is even better to predict some intrusion if it is benign than the opposite – benign but actually, it is some intrusion. On the other hand, there are also tasks, that require precision than predicting more *True* even if it is incorrect. In some tasks predicting *True* could mean making some expensive procedure. That is why model performance is measured by different metrics.

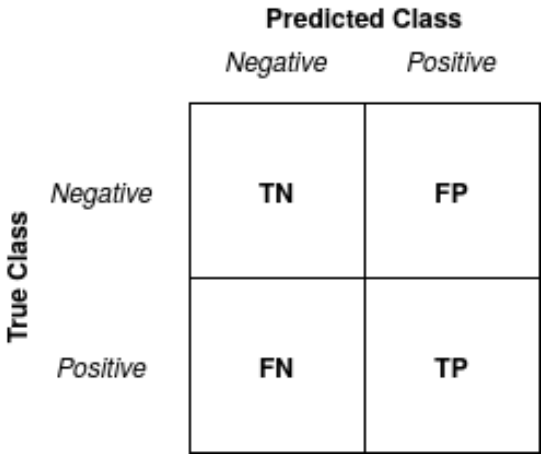


Figure 2.1: Binary Confusion Matrix

Metrics can be described on the Confusion Matrix (see Figure 2.1). Let us mention only few of them used for intrusion detection models[24]. The first metric is Recall:

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

It measures how well a model recognizes positive samples. In intrusion detection, it would be appropriate to get $Recall = 1$, because recognizing all intrusions is very important. The second metric is Precision:

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

It measures how precise predictions are. It is useful when predicting negative samples as positive is not suitable and also it is used for defining next metric. The metric is F1 Score:

$$F1\ Score = 2\frac{Precision \times Recall}{Precision + Recall}$$

It takes into account both Recall and Precision, so it gives more information about how well a model performs. The last metric is Accuracy:

$$Accuracy = \frac{TP + TP}{TP + FP + TN + FN}$$

Accuracy also gives global information about how well a model performs, but it is not very suitable for unbalanced classes. If one class has much more samples than other class and the model correctly classify only the major class, then Accuracy is high, but the model is not able to recognize the minor class. In this case, F1 Score is better than Accuracy.

The intrusion detection dataset is very unbalanced. F1 Score may be very useful in balancing Recall and Precision and to get global information about model performance.

2.2.4 Logistic Regression

Logistic Regression is a linear model. It is a simple model for binary classification, but it can be used also for multiclass classification. The prediction works in two

steps. First, similar as in Linear Regression, the linear combination is computed $w^T x$, where $x \in R^n$ is a n dimensional input column vector and $w^T \in R^n$ is a n dimensional row vector of parameters that are optimized during a training. This step is enough for regression tasks, but for classification tasks one more transformation has to be applied. The transformation is called logistic function or also known as sigmoid or softmax function. The function gives a probability distribution of how probable is that the sample x is positive. The final Logistic Regression hypothesis is defined as:

$$P(y = 1|x) = h(x) = \frac{e^{w^T x}}{1 + e^{w^T x}}$$

To find optimal parameters a loss function has to be defined.

Requirements for loss functions are to give bigger error if the prediction is incorrect and smaller error if the prediction is almost correct or correct. However, the hypothesis h produces a probability distribution and describes data likelihood, so it is not possible to only subtract a predicted class from a true class. The loss function comes from the computing data likelihood. Let data be identically and independently distributed, then data likelihood is defined as:

$$P(Y|X, h) = \prod_{i=0}^{n} P(y_i|x_i, h)$$

$$P(y_i = 1|x_i, h) = h(x_i)$$
$$P(y_i = 0|x_i, h) = 1 - h(x_i)$$

where Y and X are training labels and training data and h is a hypothesis. This raw calculation has several disadvantages and is not very appropriate to be used in this form. That is why logarithm is applied. Now the equation is easier:

$$log(P(Y|X, h)) = \sum_{i}^{n} log(P(y_i|x_i, h))$$

This form of equation describes only data likelihood, but does not say anything about what labels hypothesis should predict. One more modification needs to be done. Let us add information about true labels:

$$log(L(X)) = \sum_{i}^{n} \begin{cases} log(h(x_i)), & \text{if } y_i = 1 \\ 1 - log(h(x_i)), & \text{if } y_i = 0 \end{cases}$$

The created loss function above is well known Cross-Entropy loss[24].

Unfortunately, the training of Logistic Regression is not simple as the training of Linear Regression. Parameters have to be optimized to produce the smallest possible error. Optimization problem is defined as:

$$min_w - log(L(X))$$

It can be optimized by one of the gradient descent algorithms, which is able to find a local optimum.

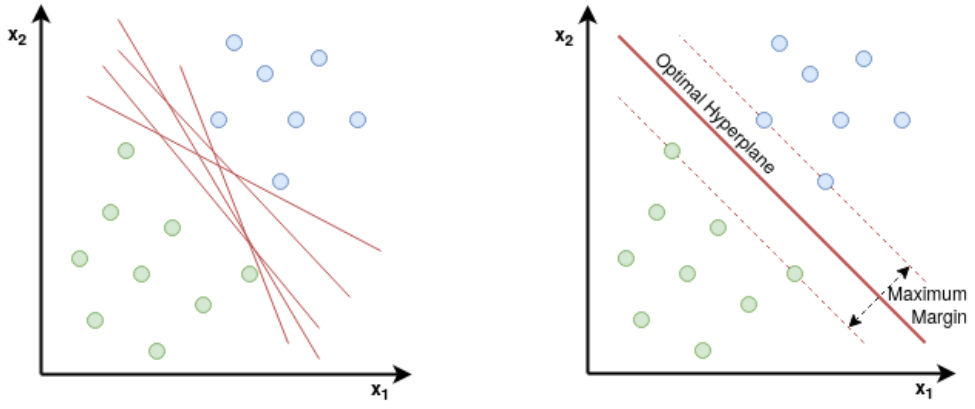
Advantages of Logistic regression are that it is simple, fast and easy to interpret. In intrusion detection, it would be better to have a fast model, because

a network communication is fast too and many services and users communicate on a network. On the other hand, Logistic Regression is simple and capable of only discovering linear dependencies. It is not bad if there are not more complex dependencies than linear dependencies. Logistic Regression may have one more usage. It multiplies each feature with its weight and the weight can be interpreted as the importance of the feature. Weights are real numbers, but if absolute value of the weight is taken, then the bigger the weight is, the more important the feature is. The training may produce really big weights and it would not help to discover the most important features, but weights can be regulate. As mentioned before, the L1 regularization technique helps to keep weights of less important features close to 0. Logistic Regression with the L1 regularization can be trained for each type of intrusion and then weights may provide a characterization of each type of intrusion. Weights may suggest to completely remove some features, because they may be unimportant. Also they may help human experts with what they should looking for in an investigation of some intrusion.

This model might not provide very good results, but can be very helpful during exploring the data and deciding what to do next.

2.2.5 Support Vector Machine

Support Vector Machine[25] (SVM) is a non-linear model. It is based on a linear model, but with two small improvements. The improvements are very important. Because of them SVM can separate even not linear separable problems. A linear separable problem is a problem, where it is possible to separate two classes with one hyperplane. One example of the linear separable problem can be two not overlapping gaussians with different mean and not very big variance. In this example, it is possible to create one hyperplane between them and separate these two clusters. Unfortunately, there are not only linear separable problems, but also other more difficult problems, so let us review SVM's improvements.



(a) Possible separating hyperplanes. (b) The optimal separating hyperplane.

Figure 2.2: Example of possible separating hyperplanes and the optimal separating hyperplane[26].

The first improvement is related to finding the best separating hyperplane and is contained in the name. In the example with two gaussians, the separating hyperplane can be created in many ways. The hyperplane can have many different

orientations and positions and still it can separate two classes very well (see Figure 2.2a). The question is which separating hyperplane is optimal and how to find it. The optimal separating hyperplane would be the hyperplane, which is far from the two classes, somewhere in the middle between them. This is done by introducing so called support vectors. Support vectors define the margin of the separating hyperplane and SVM tries to find the maximal appropriate margin (see Figure 2.2b). Support vectors are samples from the training data that influence the position and orientation of the hyperplane. However, introducing only support vectors would not help in separating not linear separable problems or noisy data, that is why slack variables are introduced. Slack variables allow to some amount of samples be in the margin of hyperplane or even to be missclassified (see Figure 2.3). It brings more variability in finding the optimal hyperplane.

Support vectors and slack variables are hidden in the final hypothesis. In comparison with Logistic Regression, SVM does not provide a probability distribution, but only $\{-1, 1\}$ predictions. The hypothesis is defined as:

$$h(x) = sign(x^T\beta + \beta_0)$$

where β and β_0 are parameters optimized during training. The training is again an optimization problem. The idea is to maximize width of margin M.

$$max_{\beta,\beta_0,\|\beta\|=1} M$$

$$\text{subject to } y_i(x_i^T\beta + \beta_0) \geq M, i = 1, ..., n$$

By adding slack variables ξ_i and using relation between β and M the optimization problem can be written as:

$$min_{\beta,\beta_0} \frac{1}{2} \parallel \beta \parallel^2 + C\sum_{i=1}^{n} \xi_i$$

$$\text{subject to } \xi_i \geq 0, y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i, i = 1, ..., n$$

where parameter C gives importance for slack variables. For example, a linear separable problem would have $C = \infty$. In next steps, the optimization problem is transformed into the Lagrangian dual objective function and becomes a convex quadratic programming problem.

This was only the first improvement, which helps with finding the optimal separating hyperplane, but does not help with separating a not linear separable problem. The second improvement is Kernel trick. The idea is if it is not able to separate two classes in the original space, then transforming samples into more dimensional space may help with their separation. However, the transformation may be slow and expensive and it is used only for scalar product. It would be great to have a function that does not need to do a transformation first, but can compute scalar product and then apply the transformation. Kernel trick is all about using such functions called Kernel functions. The formal definition is:

$$K(x, y) = \langle f(x), f(y) \rangle$$

where K is a Kernel function and f is a function, that transforms input from original space into more dimensional space. Some of the Kernel functions are d^{th}-Degree polynomial, Radial basis and sigmoid. Kernel trick has to be added into

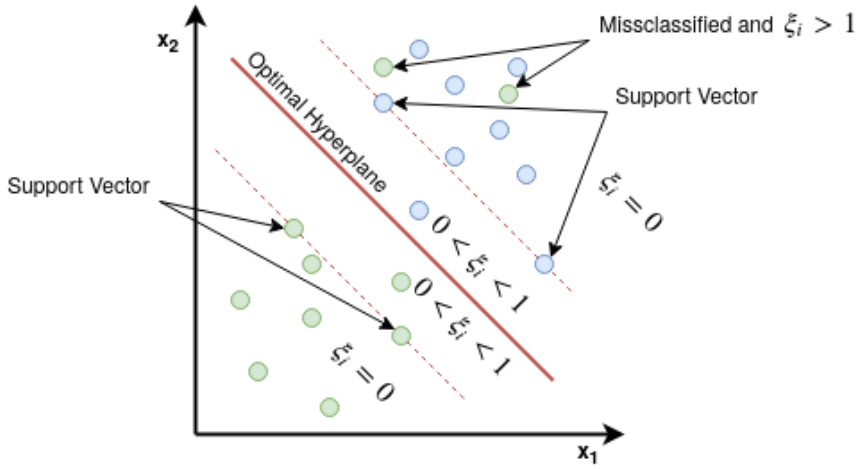


Figure 2.3: Example of SVM and its slack variables for a not linear separable problem.

the final optimization problem to train SVM with some Kernel function. SVM without Kernel trick is a linear model, but with using Kernel trick it becomes a non-linear model.

SVM model may help with more difficult intrusion detection, where Logistic Regression is too simple for that. Some intrusions may have more complex dependencies than linear dependencies. SVM can use one of the Kernel functions and discover it. On the other hand, the intrusion detection dataset is not small and, as mentioned, the training is not easy. The bigger the dataset is, the slower the training is. The training complexity is between $O(n^2)$ and $O(n^3)$, where n is size of the dataset. It depends on what method it uses for the training. In case of intrusion detection dataset, it may be a problem, because size of the dataset is 16 232 943 samples and 80 features. It may take too much time for the training and also the prediction may be slow. With growing dataset it can be solved by selecting only part of the data for the training, but it may loose some information. Again SVM can be trained for each type of intrusion. In comparison with Logistic Regression, it may be better, but also it may be impossible to train.

2.2.6 Decision Tree

Decision Tree is completely different model than previous mentioned models. Previous models try to find the optimal separating hyperplane, but Decision Tree prefers to create rectangles. It splits the whole feature space into several rectangular regions and in each region it predicts one specific value. In a classification problem, it predicts one class or probability distribution of classes. In a regression problem, it predicts one real value. The model is a simple binary tree, where in each node is one decision rule, that splits data to two parts (see Figure 2.4). Let $x = (x_1, x_2, ..., x_n)$ be a sample. Then the splitting rule[27] is defined as:

$$R_1(j, s) = \{x | x_j \leq s\} \text{ and } R_2(j, s) = \{x | x_j > s\}$$

where $s \in R$ is a suitable constant. Predictions are in leafs. In classification, it predicts the most frequent class or creates the probability distribution of classes.

In regression, it predicts the mean of values. Intrusion detection is a classification problem, that is why it is only focused on classification here.

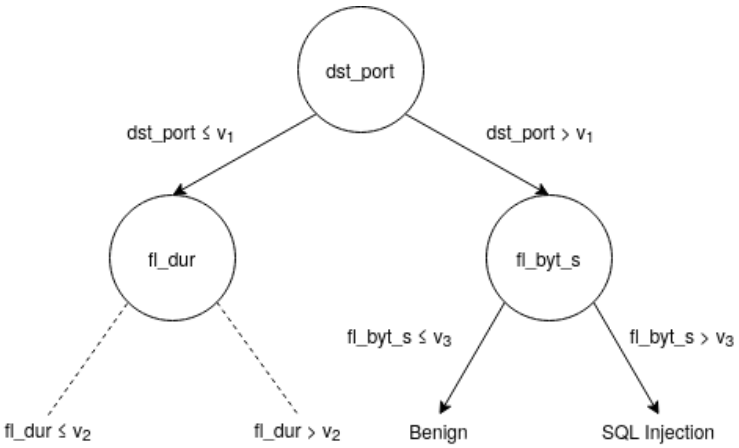


Figure 2.4: Tree Structure.

The training of Decision Tree for classification is again optimization of loss function as in Logistic Regression. The most used loss functions for Decision Tree are Gini Index and Cross-Entropy. First of all, let us defined the probability prediction in a node. Let R be a region represented by the node and N be a number of samples belonging into the region R. Then the probability of class c in the node is:

$$p_c = \frac{1}{N} \sum_{x_i \in R} I(y_i = c)$$

Then the Gini index is defined as:

$$\sum_{c \neq c'} p_c p_{c'} = \sum_{c=1}^{C} p_c (1 - p_c)$$

and Cross-Entropy is defined as:

$$-\sum_{c=1}^{C} p_c log(p_c)$$

Now the growing of a tree is simple. In each node, it goes over all available features and tries all possible splitting points. For each splitting point, it computes the loss function and remembers only the splitting point that best minimizes the loss function. At the end, it selects the splitting point that best minimizes the loss function. Then the growing continues in the left branch and the right branch until a stop condition is met. The stop condition can be that the minimization is very small and does not exceed some threshold, but it is not very useful. Usually, a large tree is grown and then pruning techniques are applied. The disadvantage of a large tree is, that it may overfit, but a small tree may underfit, that is why tuning of the size is tested on a validation set.

Decision Tree is very a simple model, but main advantage is that it is human-readable. For example, human experts could see how the model classifies some intrusion. It is possible to see what features and how exactly they are used in the decision. Also tree can be used for creating rules. Decision Tree is simpler than SVM, but it may produce better results than SVM. Overfitting may be a bigger problem than underfitting.

2.2.7 Rules

There are many methods how to create rules for classification. This work uses the method described by Jerome H. Friedman and Bogdan E. Popescu[28][29]. Their method extracts rules from Decision Tree ensembles. The Decision Tree can be seen as a set of rules. Each path from the root of the tree to some leaf is a rule and also each path from the root to some node is a rule. These rules can be easily extracted from the tree and transformed into a logical formula with conjunctions. To get short and diverse rules, the method uses Decision Tree ensembles. Ensemble methods create a big number of small trees. It usually creates tens or hundreds of trees. Each tree is trained on different subset of the training set, so each tree may be completely different. The tree's height is controlled by the parameter. This is very useful, because it is possible to control the maximal length of rules.

Let us defined the rule. Let $x = (x_1, x_2, ..., x_n)$ be a sample and $s_i = (a, b\rangle$ be an interval of values, where i is the $i\text{-}th$ feature. Then the rule can be defined as:

$$r(x) = \prod_{i=1}^{n} I(x_i \in s_i)$$

and $r(x) \in \{0, 1\}$. Rules are capable of binary classification only. The rule may not contain s_i for each feature i. The features not specified by s_i can have any value. Now let M be a number of trees and t_m be a number of leafs in $m\text{-}th$ tree, then the total number of rules is:

$$\#rules = \sum_{m=1}^{M} 2(t_m - 1)$$

The number of rules grows quickly with the increasing number of trees, but not every rule is useful. Duplicates and not very accurate rules have to be removed. Skope-Rules[30], which is Python library implementing [28] and [29], uses Out-of-Bag error to select appropriate rules. After selecting the appropriate rules, there is one more postprocessing. The postprocessing goes over all selected rules and removes all rules, that have smaller precision or smaller recall than some threshold. The threshold of minimal precision and minimal recall can be passed as a parameter.

The advantage of this method is that it is based on existing fast algorithms. There are many good algorithms for creating ensembles also there are many algorithms for training Decision Trees. The Decision Tree and ensemble methods perform well, so it may provide good rules.

In the intrusion detection problem, it may bring good results. Decision Trees may provide many different rules and by defining the maximal height of trees it is possible to control the length of rules. It also provides the possibility to tune precision and recall of rules.

2.3 Unsupervised Learning

As mentioned, the unsupervised learning[31] operates with unlabeled data. It does not search for any optimal hypothesis. It tries to find similarities and relations in the data. One of the unsupervised learning methods is clustering. The

clustering groups similar samples into one cluster. The similarity is usually defined as Euclidean distance. The clusters can be used for detecting possible classes or some model training. Other unsupervised learning method is dimensionality reduction. The dimensionality reduction is useful in reducing data complexity. It transforms data samples from high dimensional space into low dimensional space, but it tries to preserve similarities and relations as much as possible. The unsupervised learning is used for the data visualization, so let us now discuss some methods for dimensionality reduction.

2.3.1 Principal Component Analysis

Principal Component Analysis[32][24] (PCA) is a transformation that tries to maximize the data variance. The data are projected into so called principal components. The principal components are sets of data projections with several properties[31]. The first property is that principal components are ordered by variance. The first principal component has the biggest variance and the last principal component has the lowest variance. The second property is that they are mutually uncorrelated. The PCA transformation has simple formula. Let $X \in R^{m \times n}$ be data samples, where samples are column vectors, and $W \in R^{m \times k}$ be a transformation matrix. Number n defines a number of samples and numbers m and k define a number of dimensions of the original space and new space. The number k is usually a much lower than m. The transformation is defined as:

$$Y = W^T X$$

where $Y \in R^{k \times n}$ are data projections. The values of column vectors in Y are linear combinations of the original features, so the original features are not lost but hidden in the values of the column vectors. The problem is to find the optimal transformation matrix W.

As mentioned in the beginning, PCA tries to maximize the data variance, so the finding of principal components can be defined as:

$$max_W Var(Y)$$

The computing of $Var(Y)$ is nothing more than product of W and covariance matrix $Cov(X)$:

$$Var(Y) = Var(W^T X) = E[(W^T X - E[W^T X])^2] = W^T Cov(X) W$$

However, the matrix W can have very big values, so constraint $\|w_i\| = 1$, $\forall\ i = 1, ..., m$ is added. The final result of the maximization problem for the vector w_1 is the eigen vector of the covariance matrix $Cov(X)$ with the largest eigen value. The second vector w_2 is obtained in similar way, but the vector has to be orthogonal to w_1. It again gives the eigen vector, but the vector has the second largest eigen value. The transformation matrix consists of m first eigen vectors ordered by their eigen values. The last modification is centering the data before their transformation:

$$Y = W^T (X - \mu^T \mathbf{1})$$

where μ is the mean of the data X and $\mathbf{1}$ is a row vector of ones.

In addition to the transformation, PCA can provide explained variance ratio. The explained variance ratio is the ratio between the principal component variance and the data variance. It says how well the data are described by the principal components. The value 1 says that there is no difference between the data and the principal components. The values closer to 0 says that the principal components do not describe the data well. The ratio helps with choosing the appropriate number of the principal components to not lose a lot of information about the data.

2.3.2 Multidimensional Scaling

Multidimensional Scaling[31] (MDS) is different than PCA. It transforms the data samples from the high dimensional space into low dimensional space, but it tries to preserve the distances between the data samples. The distance is usually measured as Euclidean distance $d_{ij} = \|x_i - x_j\|$, where x_i, x_j are the data samples and d_{ij} is the distance between them. MDS can be expressed as:

$$Y = W^T X$$

where W, X and Y are the same as in PCA. Now there is only problem how to find the transformation matrix W. PCA solves this as maximizing the variance of the transformed data. However, MDS uses only the distances. It does not need anything more. MDS defines the finding of optimal transformation matrix as the minimizing the so called stress function:

$$min_W S, \text{ where}$$
$$S(y_1, y_2, ..., y_n) = \sum_{i=i'} (d_{ii'} - \|y_i - y_{i'}\|)^2$$

This is know as Kruskal-Shephard stress function. It tries to preserve the pairwise distances as well as possible. This is not the only variant of stress function. Another stress function is Sammon stress function:

$$S(y_1, y_2, ..., y_n) = \sum_{i=i'} \frac{(d_{ii'} - \|y_i - y_{i'}\|)^2}{d_{ii'}}$$

The Sammon stress function puts more emphasis on smaller pairwise distances. Both functions can be optimized by a gradient descent algorithm. MDS may give good result, but the problem is the number of pairwise distances. The number of pairwise distances grows very quickly with the growing number of samples. It grows quadratically, so it may be a problem for large data.

2.3.3 Locally Linear Embedding

Locally Linear Embedding[31] (LLE) is similar to MDS, but as the name suggests it aims at the local distances. It tries to preserve the local structures. First, it defines the sample neighbors. The sample neighbors are the closest samples measured by Euclidean distance. Neighboring can be solved by K-nearest neighbors $N(i)$, where $N(i)$ are the neighbors of a sample x_i. The number of neighbors is passed as parameter of LLE. Now the minimization problem can defined as:

$$min_{w_{ik}} \|x_i - \sum_{k \in N(i)} w_{ik} x_k\|^2$$

where $w_{ik} = 0$ for $k \notin N(i)$ and $\sum_{k=1}^{n} w_{ik} = 1$. After finding the optimal w_{ik}, the w_{ik} is fixed and the final transformation starts. The last minimization problem is defined as:

$$\sum_{i=1}^{n} \|y_i - \sum_{k=1}^{n} w_{ik} y_k\|^2$$

Each dimensionality reduction method may give different results. It does not mean that one method works best for all data. It always depends on the data. Let us now have a look on the last method that uses a probability distribution for describing distances between the data.

2.3.4 t-Stochastic Neighbor Embedding

The method t-Stochastic Neighbor Embedding[33] (t-SNE) is improved variant of Stochastic Neighbor Embedding (SNE). Instead of operating with Euclidean distances, it uses a probability distribution that defines the similarity in the data. The similarity measure is defined as the conditional probability $p_{j|i}$. The probability $p_{j|i}$ says how much it is probable that the sample x_i has the sample x_j as its neighbor with respect to Gaussian distribution with center x_i. However, computing of Gaussian distribution for x_i needs its variance σ_i. The variance σ_i is computed by binary search algorithm and Perplexity given by user.

The probability distribution defines the similarity measure very well in high dimensional space. The same has to be done in low dimensional space, but in low dimensional space the variance σ_i is fixed to $\frac{1}{\sqrt{2}}$. Let $q_{j|i}$ be a probability in low dimensional space. It needs the similarities only between the x_i and x_j, where $i \neq j$, so the probabilities for $i = j$ are defined as $p_{i|j} = 0$ and $q_{i|j} = 0$. There are two probability distributions now. The first P_i describes the similarities in high dimensional space and the second Q_i describes the similarities in low dimensional space. The objective is to get Q_i similar to P_i as much as possible. If Q_i is similar to P_i, then the data in low dimensional space are similarly distributed as in high dimensional space.

It is again an optimization problem, but it needs a function comparing two probability distributions. One of the functions is Kullback-Leibler function that is similar to Cross-Entropy loss:

$$L = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} log \frac{p_{j|i}}{q_{j|i}}$$

The only thing now is to apply gradient descent algorithm to optimize the loss L. However, this problem is difficult to optimize.

The method t-SNE improves these difficulties. One of the simplifications is using of symmetric SNE. It defines new probabilities p_{ij} and q_{ij}, where $p_{ij} = p_{ji}$ and $q_{ij} = q_{ji}$ $\forall i, j$. The probability q_{ij} is computed in the same way as in SNE but p_{ij} is defined as:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$$

This simplifies the gradient of the loss L, so it is easier to optimize. The second problem is so called Crowding Problem[34]. The problem is solved by replacing the Gaussian distribution in low dimensional space with Student's t-distribution

with one degree of freedom. The benefit of Student's t-distribution is its behavior far from the center.

The method t-SNE works completely different than the previous methods, so it may provide interesting results in the data visualization, but its computation is not cheap.

3. Model Training

This chapter presents the process of the model training from the data preparation, feature selection and choosing a model for the final training. Section 3.1 describes the data preprocessing. It focuses on the data transformation, removing useless features and how to deal with the large dataset on a personal computer. Section 3.2 presents the data visualization. Various techniques are tested and the best result is attached in this section. Section 3.3 discusses the feature selection. The dataset has 80 features and not every feature may be important. It is not necessary to keep each feature in the dataset. The section discusses several methods and compares their results. The chapter ends with Section 3.4 presenting results of various models, their advantages and disadvantages in the intrusion detection problem.

3.1 Data Preprocessing

The whole dataset is stored on Amazon Web Services (AWS). To download the dataset, it requires to have AWS Commad Line Interface (CLI)[1], that allows to access AWS and manipulate with the data. As mentioned in the description of the dataset (Chapter 1), the dataset contains more than the prepared data for a model training. It contains also raw data. To download the whole dataset, run:

```
> aws s3 sync --no-sign-request --region your-region
  "s3://cse-cic-ids2018/" dest-dir
```

where *your-region* is AWS region[2] and *dest-dir* is a directory, where it should be downloaded. However, the whole dataset has more than 400 GB. If the raw data are not needed, it is unnecessary to download the whole dataset. For listing files and directories on AWS, run:

```
> aws s3 ls --no-sign-request --region your-region
  "s3://cse-cic-ids2018/" --recursive --human-readable --summarize
```

Based on results of above command, it is possible to download only specific part of the dataset. In our case, only CSV files are important:

```
> aws s3 cp --no-sign-request --region your-region
  "s3://cse-cic-ids2018/Processed Traffic Data for ML Algorithms"
  dest-dir --recursive
```

After downloading is completed, nine CSV files can be found in the directory *dest-dir*. Each file represents one day of intrusion simulation. These files needs to be put together into one big file.

If CSV files have the same header, it is easy to put them together into one big file. However, one file has different header than the others and also the headers are contained in the files more than once. The file recorded on Thuesday 20.02.2018

[1] AWS CLI documentation and download link can be found on `https://aws.amazon.com/cli/`.

[2] AWS region list can be found on `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html`.

has four more columns than the others. The columns are *Flow ID*, *Src IP*, *Src Port*, *Dst IP*. These columns can be removed, because *IP* and *Flow ID* describe communication in the specific network, but the model should be applicable in any network. Also *Src Port* can be omitted. After this, all files have the same header. Now the files can be put together. Headers that are contained in a middle of files can be skipped.

One of the most used programming languages for machine learning is Python[3]. Everything in this work is implemented in Python. Python has many libraries for loading and manipulating with CSV files. One very often used library is Pandas[4]. It allows to compute basic statistics, drop columns, join more files into one big file and much more. Unfortunately, it is implemented for small data. The bigger data are, the more difficult it is to use. Also every experiment is run on machine with only 12 GB of RAM and 16 GB of Swap. Pandas consumes more than 12 GB of memory and the loading of the data is very slow. Other options are to use PySpark[5] or to create own data transformer. PySpark is very good tool for Big Data, but it is more useful for distributed computing than for running on a single machine. For running it on the single machine it is too heavy, so creating own data transformer may be a better way.

There is a need to analyze data, transform the data and then visualize them. The analysis should provide basic statistics such as minimum, maximum, mean, standard deviation, number of unique values, number of distinct values, most frequent values and number of values satisfying some condition. It must not load all data into RAM. Each statistic is represented in Python by one class, which extends the abstract class Analyzer. Each class is capable of analyzing row by row and joining more results into one final result. The Analyzers are used in the class AnalysisRunner, which is capable of running the whole analysis and storing the results into CSV files. The class AnalysisRunner can be run in a single thread or in multiple threads. It processes the data row by row. The whole analysis can be configured by YAML[6] file. The example of yaml configuration is:

```
- ACK Flag Cnt: &id001
  - Min
  - Max
  - Mean
  - Unique
  - Distinct
  - StandardDeviation
  Active Max: *id001
  Active Mean: *id001
  Active Min: *id001
  Active Std: *id001
  Bwd Blk Rate Avg: *id001
  Bwd Byts/b Avg: *id001
  Bwd Header Len: *id001
```

Data transformations are implemented in the class Transformer. The class Transformer allows to transform values, drop columns or replace some value by another

[3] `https://www.python.org`

[4] `https://pandas.pydata.org/`

[5] `https://spark.apache.org/docs/latest/api/python/index.html`

[6] YAML specification and projects can be found on `https://yaml.org/`.

value. It again processes the data row by row. The data visualization is done by another Python library called Matplotlib[35]. Matplotlib[7] is one of very popular libraries for creating various type of graphs.

The first analysis is run to discover classes ratio and invalid or missing values. Missing values are not found, but there are invalid values or values not usable for machine learning. Features that contain invalid values are *Flow Byts/s* and *Flow Pkts/s*. Invalid values are *inf* and *nan*. There are two options – replace values by suitable value or remove incorrect samples. Here invalid values are replaced by suitable values as Qianru Zhou and Dimitrios Pezaros did in their work [6]. The value *inf* should refer to some very big number, that is why *inf* is replaced by a maximum. The value *nan* is hard to say, what it should be. It could be some incorrect division or just some issue during measurement. It can be replaced by 0 or by a most frequent value. The value 0 would suggest, that there were 0 somewhere in the computing the feature. On the other hand, it may be completely different reason why *nan* is there. After all, the most frequent value appears as the most suitable value, because it does not bring any strange activity. The final replacements are:

- *Flow Byts/s*:
  - *inf* is replaced by 1806642857.14286
  - *nan* is replaced by 0
- *Flow Pkts/s*:
  - *inf* is replaced by 6000000.0
  - *nan* is replaced by 1000000

The second analysis computes basic statistics. First, it is computed for the whole dataset and then for each class separately. The analysis discovers that some features have only one distinct value and the others have a big range of values. The features with only one distinct value are *Bwd Blk Rate Avg*, *Bwd Byts/b*, *Avg, Bwd PSH Flags*, *Bwd Pkts/b Avg*, *Bwd URG Flags*, *Fwd Blk Rate Avg*, *Fwd Byts/b Avg*, *Fwd Pkts/b Avg*. These features can be removed, because they does not bring any useful information. Also *Timestamp* can be removed, because time is included in statistics of flow such as minimum, maximum, mean and standard deviation. Most of the other features have very large standard deviation that is caused by the very big range of values. These big ranges of values may influence the training and make it more difficult. It is better to normalize these features by some normalization method before the model training.

Before normalizing the data, it has to be split to a training set and test set. The training set consists of 80% data and the test set consists of 20% data. The test set is left for the evaluation of the final model. The training set is then divided on a smaller training set and validation set. The validation set consists of 20% data from the training set and the rest is the smaller training set. Now there is a question how to choose the data. The data can be chosen randomly or in some order. The data are time series, so it is more appropriate to start at beginning and stop at some point. The first 80% of the data is the training

[7]Documentation and examples can be found on `https://matplotlib.org/`.

set and the rest is the test set, but the classes ratio is preserved in both sets. Similarly it is done for the validation set. The data normalization is done after removing useless features and splitting the data. The data are normalized in the usual way by standardization.

The problem is still the size of the data. Some amount of RAM takes the model and some models need to have the whole training set in RAM. After the data normalization, values are small float number, but with high precision. The precision can be decreased, so there will not be too long numbers. Next step is to convert the data into *numpy array*. *Numpy array* uses less memory, but it would be better to have in memory only the samples, that are required. This lazy loading is allowed by *numpy's memory-map*. *Numpy array* is stored as binary file and only the rows or columns that are required are loaded into RAM.

3.2 Data Visualization

In addition to computing the statistics, box plots are created. Box plots display 6 values – minimum, maximum, first quartile, median, third quartile and outliers. It is one of ways exploring the data distribution. Some of interesting box plots are attached at the end of this work (see Attachment A.1). On box plots can be seen many things. First of all, they explain why standard deviation is so large. Some features have many large or small outliers, which make standard deviation bigger. Standard deviation is related to skewness. They imply that some features may have some skew probability distribution, because median of values is not in the middle but is shifted to left or right. The last thing is that some classes differ a lot and may be easily separated. Some features may appear as important in the classification.

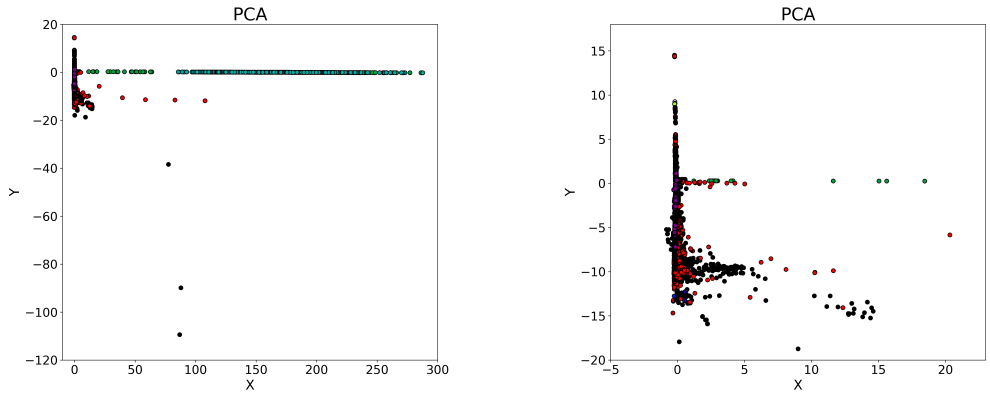
It is also possible to visualize samples of the dataset. However, samples have many dimensions. After removing useless features, there is still 70 features. The samples dimensions have to be reduced before visualizing. The data visualization has to have samples with only 2 dimensions. It is possible to visualize samples with 3 dimensions, but 2 dimensions are usually enough. There are many methods for the dimensionality reduction. Most of the methods are implemented in machine learning library called Scikit-Learn[36]. The dimensionality reduction is tested by 4 methods. Let us start with less successful methods.

The first is Multidimensional Scaling (MDS). MDS is a method, that tries to preserve distances between the data points during the dimensionality reduction. Preserving distances between the data points may provide good results, but the method is not optimized for large data. The dataset is reduced on reasonable size, that is still representative, but MDS is not able to finish its computing on such data.

The second method is Locally Linear Embedding (LLC). This method does not try to preserve distances between all data points but only between the data point and its neighbors. The number of neighbors is a parameter, that can be tuned to get some good result. It looks more promising than computing distances between all data points. However, even LLC is not able to finish its computing.

One of newer methods is t-Stochastic Neighbor Embedding (t-SNE). It works differently than the previous methods. Instead of computing Euclidean distances, it uses the probability as the measure of similarity. On well known examples it

gives better results than other methods, but the computation is not very easy. As expected, this method fails on such large data. The similar method is Uniform Manifold Approximation (UMAP), that should work better than t-SNE. Unfortunately, UMAP is not able to give any result too.



(a) The graph visualizes all transformed data.

(b) The graph is zoomed in the cluster area to show the cluster details.

Figure 3.1: The graphs show the first two principal components of PCA. The x axis represents the first principal component and the y axis represents the second principal component. Only part of the data samples is transformed by PCA to reduce the computation complexity.

The only method that works good is Principal Component Analysis (PCA). PCA tries to maximize the data variance. The bigger variance, the more data are spread in a space. The PCA computation is not difficult. It works with the covariance matrix and its eigen vectors. It only finds the eigen vector with the largest eigen value to create the first principal component. To create second principal component it takes the eigen vector with the second largest eigen value etc. Moreover in Scikit-Learn, they implemented incremental PCA, which allows to work with batches and does not need the whole data in RAM. The result of PCA can be seen in Figure 3.1a and 3.1b. The first graph demonstrates first two principal components of the reduced data. Classes of the data points are distinguished by color. It shows that some class is completely outside of the cluster, but other classes are in one cluster. The second graph is focused only on the cluster, but even zooming does not suggest any good separation of classes. It means that the first two principal components may not be enough to describe the data well or the data may be difficult to separate.

PCA does not provide only the transformed data points but also the explained variance ratio for each principal component. The bigger explained variance ratio is, the better data are described by the principal component. The explained variance ratio can be seen in Figure 3.2. It reveals, that the first 30 components are enough to describe the data well. For example, 31 components have ratio 0.983, so there is very small difference between the variance of 31 components and the original data variance. This can be used to reduce dimensions of the data and train the model on less features. The training is then faster and the data are smaller. However, not every feature may be important for the model.

There may be only few features important for the model and the dimensions of the data may be reduced more than by PCA. Moreover, the principal components are combinations of the original features. It is not very suitable for training short and simple rules, where getting rules for original features is more appropriate. However, for some model it might be good. Let us have a look on feature selection.

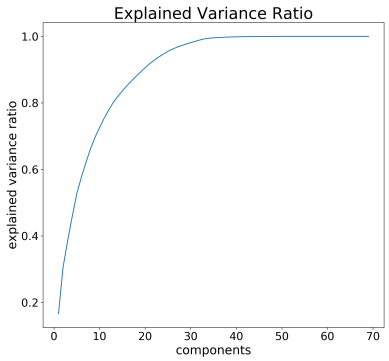


Figure 3.2: The graph shows the explained variance ratio. The explained variance ratio reaches value 1.0 around the first 30 principal components. The first 30 principal components may be considered sufficient.

3.3 Feature Selection

Feature selection tries to find the most important features and select them for the model training. Feature importance can be defined in various ways. For example, the Decision Tree has different feature importance measure than the Logistic Regression. Each model may consider different features as important, but if some feature almost defines the class of samples, it should have high importance for each model. Not every model is able to provide feature importance. In this work, two models are used for feature importance – Logistic Regression and Decision Tree. In the Logistic Regression, feature importance is extracted from its weights $w = (w_1, w_2, ..., w_n)$ and in the Decision Tree feature importance is defined as normalized total reduction of criteria by the feature. Feature importance may help in discovering classes characterization. The feature importance is computed for each class separately. For each class both models are trained and then feature importance is extracted. Let us have a look on the Logistic Regression.

The Logistic Regression is trained on the training set without the validation set. The dataset has 15 classes, so to get feature importance for each class it has to be trained for each class. The problem is very unbalanced classes. For example, SQL Injection has only tens of samples. It does not mean it is not enough, but it may need a help with increasing its weight. Fortunately, the Logistic Regression in Scikit-Learn allows to specify the weights of classes, also it provides a method to compute their weights based on their occurrences. This persuades the model, that both classes are equally important. After tuning the weights of classes, the Logistic Regression is able to recognize most of the intrusions and the feature importance can be extracted. The features are sorted by its importance and only the names of the features are mentioned (see Attachment B.1). The results

of Logistic Regression reveal, that only small part of the features is important. With the feature selection, it may be possible to get less features than with the dimensionality reduction, but it has to be tested by cross validation or other validation method.

The Decision Tree is trained in the similar way as the Logistic Regression, but the Decision Tree does not need to tune weights of classes a lot. It achieves better results than the Logistic Regression. Again it is trained as a binary classifier, for each class separately. However, the Decision Tree requires only a few features to correctly classify the intrusions. The extracted feature importance can be found in Attachment B.1. For some intrusions it considers only 1-3 features as important. In comparison with the Logistic Regression, it is much more better.

Feature importance can be computed not only with ML models, but also with statistical methods. One of the methods is Analysis of Variance known as ANOVA. It has different forms. This work uses one-way ANOVA test. It computes F-value and P-value for each feature and based on its values it is possible to determine whether the feature is important or not. The advantage is that feature importance is not tied with any specific model and is based only on statistics. However, it has important assumptions that must be satisfied otherwise it may not be so powerful. The results of ANOVA can be seen in Attachment B.1. It provides similar results as the Decision Tree. Let us now summarize all results.

Bot Intrusion

Botnet intrusion infects some amount of machines and then it requests screenshots every 400 seconds from them. The requesting screenshots should increase amount of data sent to attacker. The most important features show, that many features describe data flow in backward direction. The size of packets in backward direction may be very important, because its standard deviation, mean and maximum appear as important. In addition to the size of packets in backward direction, there are other features describing the size of packets. All the important features may be characteristics of Botnet Intrusion.

Brute Force -Web and -XSS

In Brute Force -Web intrusion, the methods consider important the features describing the size of packets in backward direction. However, there is not any feature appearing in every method. The Logistic Regression and ANOVA have similar the most important features, but the Decision Tree has completely different the most important features. Brute Force -XSS looks similar as Brute Force -Web, but Brute Force -XSS has one more different the most important feature. The feature is the size of packets in forward direction. It is considered by the Logistic Regression and ANOVA.

DDOS Intrusions

DDOS intrusions are based on overwhelming or flooding some machine. It also appears in the feature importance. All three methods agree on features, that describe sent data in forward direction. Only DDOS attack-LOIC-HTTP has

different the most important features. In DDOS attack-LOIC-HTTP is the most important backward direction. The feature describing number of bytes sent in initial window in the forward direction is probably the most important feature of DDOS attack-HOIC. The feature appears in all three methods and ANOVA even considers it as the only important feature. Also the Decision Tree considers only two features as important. The feature describing number of packets with at least 1 byte of TCP data payload in the forward direction is probably the most important feature of DDOS attack-LOIC-UDP. It again appears in all three methods and the Decision Tree considers it as the only important feature. The common most important feature in DDOS attack-LOIC-HTTP is standard deviation of the size of packets in backward direction.

DOS Intrusions

DOS intrusions are similar to DDOS, but are led by a single machine, so this may make some difference. In each DOS intrusion, all three methods consider some feature as the most important feature. However, DOS intrusions do not differ from DDOS intrusions very much in the most important features. Some of DOS intrusions are more characterized by data flow in backward direction, but also some of them are more characterized by data flow in forward direction. However, there can be found some important features that are not considered in DDOS intrusions.

FTP and SSH Brute Force

All three methods does not consider many features as important in FTP Brute Force intrusion, but also they does not agree on the same features. The most promising features may be the features describing number of backward packets per second and minimum segment size observed in the forward direction. In SSH Brute Force intrusion, it is more difficult. The Decision Tree prefers features about header, but these features do not appear in ANOVA and the Logistic Regression. They rather prefer features describing number of packets with URG and minimum segment size observed in the forward direction.

SQL Injection and Infiltration

SQL Injection and Infiltration are difficult to recognize. It is demonstrated by how many features the methods consider as important and still the models are not able to recognize all SQL Injections and Infiltration. However, in comparison with how these intrusions behave and what they do it is difficult to detect. For example, it is usual to run some SQL statements behind the web page and then return their results. The features do not contain any specific information about SQL statements, so they may not have enough information or the information that would be more useful.

3.4 Training

The main purpose of a model training is to discover dependencies in the data and to have some baseline for a comparison with rules. In the training stage, several

models are trained to discover various dependencies and to find the best fitting model. It starts from a simple model to a more complex model.

First, the Logistic Regression is tested. It is used for both feature importance and linear dependencies testing. For each intrusion, one Logistic Regression is trained. Here it is used as a binary classifier, but there are methods how to put binary classifiers together and create the multiclass classifier. The methods are for example One-vs-All or One-vs-One. As mentioned in the previous section, the Logistic Regression has a problem with unbalanced classes. However, it can be solved by balancing the weight of classes. Unfortunately, even this does not help to get results as Qianru Zhou and Dimitrios Pezaros[6] achieved. It does not mean that there are not any linear dependencies, but there are probably some more complex dependencies. Sometimes the Logistic Regression works better for the logarithm of the data. The data are also normalized by the logarithm, but the Logistic Regression performs worse than for the data normalized by standardization. The Logistic Regression achieves in the binary classification Recall closed to 1.0 in all intrusions except SQL Injection and Infiltration. It is not a big problem to persuade the Logistic Regression to detect a specific intrusion, but the problem is F1 Score. In most intrusions it has a big amount of FN and FP. The results for binary classification are attached in Attachment B.2. The Logistic Regression is too simple model for classifying such problem. It can be also seen in Figure 3.3, that demonstrates the performance of multiclass classification. The best and usable results are for DDOS attack-HOIC, DDOS attack-LOIC-UDP, DoS attacks-Hulk and FTP-BruteForce. However, more complex models should have better results.

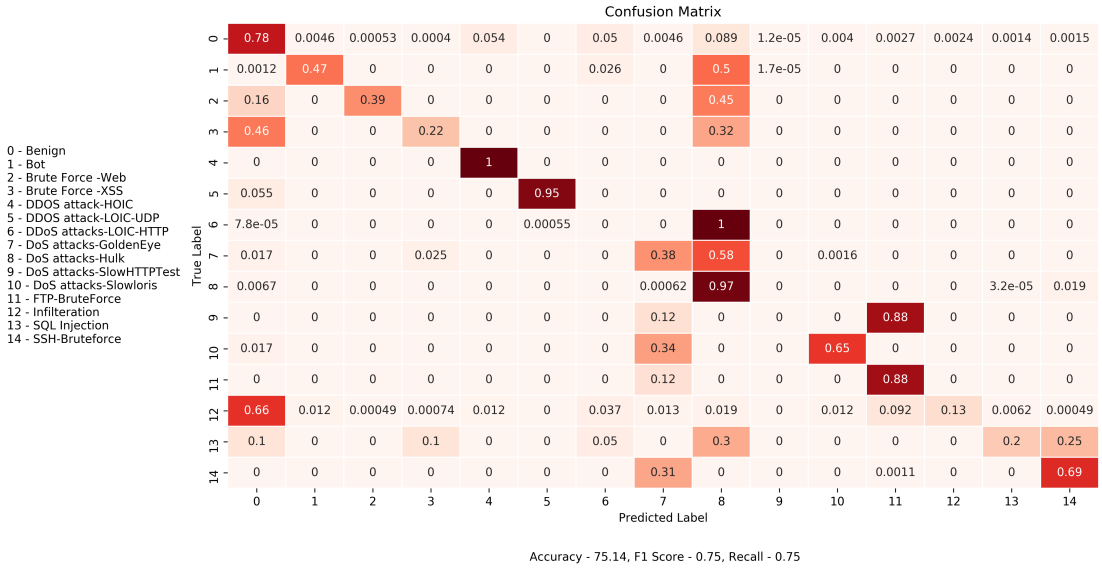


Figure 3.3: Confusion matrix for multiclass Logistic Regression (One-vs-All).

One of more complex models is Random Forest. It is basically an ensemble of the Decision Trees. It is used also by Qianru Zhou and Dimitrios Pezaros[6]. In their work, it achieves better results than the Logistic Regression. In this work, it is used differently. Instead of using it as a binary classifier it is used as a multiclass classifier. In comparison with results in [6], it achieves very similar results, but takes much more time for training than the Logistic Regression. The

worst classified intrusions are Brute Force -XSS, DoS attacks-SlowHTTPTest, Infiltration and SQL Injection. Based on confusion matrix in Figure 3.4, there may be some similarities between intrusions. For example, Brute Force -XSS and Infiltration may be very similar to Benign, because most of the positive samples are classified as Benign. Also SQL Injection may be similar to Benign, but not too significantly. One more interesting similarity may be between DoS attacks-SlowHTTPTest and FTP-BruteForce. This similarity may be also demonstrated by their feature importance (see Attachment B.1). They have similar the most important features. On one hand, the training of the Random Forest is slow, but on the other hand the classification is fast and more accurate than the classification by the Logistic Regression. The Random Forest may be a good candidate for an intrusion detection system.

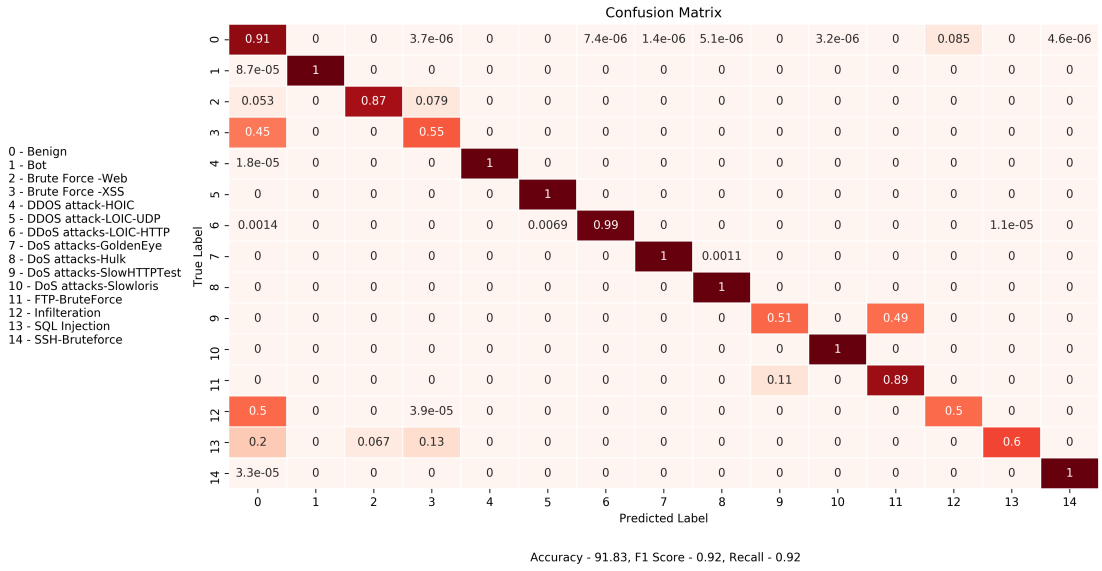


Figure 3.4: Confusion matrix for multiclass Random Forest with 200 trees.

The Random Forest has problems to classify some intrusions, but other model might be better. SVM is next model, that is more complex than the Logistic Regression. It has advantage that it can use different kernels and it may discover other dependencies. Unfortunately, the disadvantage is very slow training and very big consumption of memory. The dataset has to be very drastically reduced to be able to train SVM. First, only 1 000 000 samples from Benign samples is selected and then it is more reduced by selecting less than 1 000 000 samples, but with preserving the classes ratio except Benign class. Some information might be lost by selecting only part of the data, but still it may be representative enough. The SVM is trained with two different kernels – Polynomial with degree of 3 and RBF. The best results for multiclass classification are presented in Figure 3.5. The multiclass classifier is created by One-vs-All method. In comparison with the Logistic Regression, F1 Score and Recall is higher than for the Logistic Regression, but it is able to detect only few intrusions and others are classified as Benign. It might be caused by very unbalanced classes or the kernels may not be suitable for such problem. In comparison with the Random Forest the SVM is worse. However, the Random forest has difficulties to detect Brute Force -XSS, DoS attacks-SlowHTTPTest, Infiltration and SQL Injection. The SVM can discover

if there are more complex dependencies. The results of the SVM as a binary classifier are attached in Attachment B.3 and B.4. The best results for Brute Force -XSS and Infiltration are achieved with Polynomial kernel and for DoS attacks-SlowHTTPTest and SQL Injection are achieved with RBF kernel. In comparison with the Random Forest, the SVM does not provide better results except DoS attacks-SlowHTTPTest. However, the SVM is used as a binary classifier and the Random Forest as a multiclass classifier, so in general the SVM is worse than the Random Forest (see Figure 3.4 and 3.5). On the other hand, SVM discovers that RBF kernel may deal with some intrusions better.

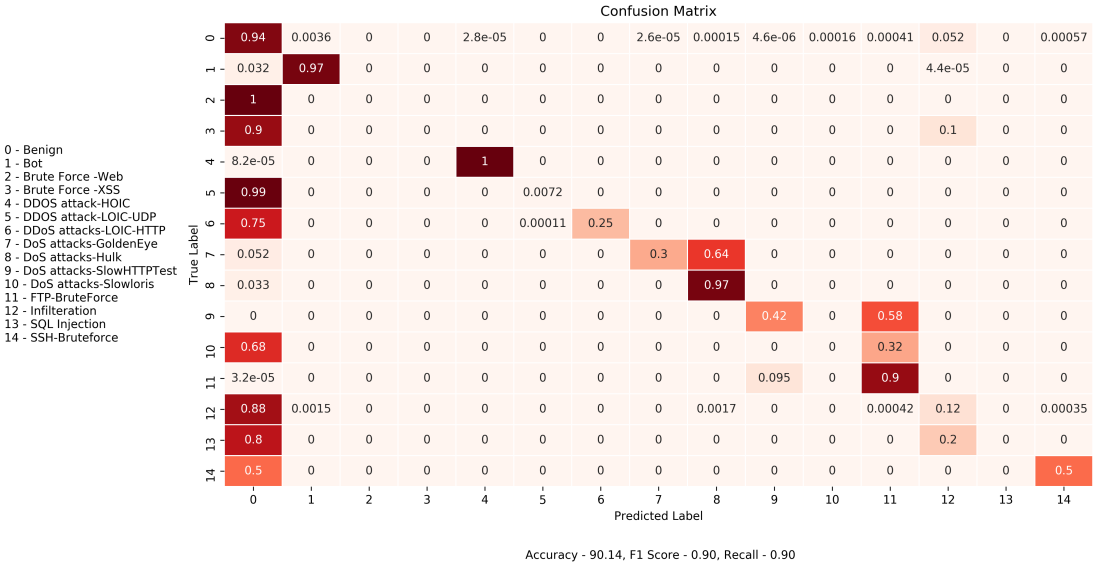


Figure 3.5: Confusion matrix for multiclass SVM with RBF kernel (One-vs-All). The SVM is trained on small part of the data with size 827 651 samples. The small part of the data tries to preserve classes ratio as much as possible with respect to a minimal number of samples for one class and except Benign class.

Unbalanced classes can be overcome by a boosting method. The boosting method is in a family of ensembles. It uses weak learners. The weak learner is a classifier, that is better than a random guess, but usually not very much. The training is iterative. Every next weak learner tries to correct the previous weak learner. Here is used AdaBoost (Adaptive Boosting). AdaBoost increases importance of wrongly classified samples, that pushes the weak learner to focus on these samples. As a weak learner is usually used the Decision Tree with the limited height. However, AdaBoost does not perform well. The result is presented in Figure 3.6.

The last trained model is Decision Tree. It is used as a binary classifier and trained for each intrusion. The training is fast, but needs balancing of classes by providing their weights. The Decision Tree achieves the best results among mentioned models (see Attachment B.2). It also achieves the best result in the multiclass classification. The result is presented in Figure 3.7. In comparison with the Random Forest, the performance significantly differs in detecting Brute Force -XSS, DDoS attack-LOIC-UDP, DoS attacks-SlowHTTPTest, FTP-BruteForce, Infiltration and SQL Injection. The Random Forest is better in detecting DDoS attack-LOIC-UDP, FTP-BruteForce, Infiltration and SQL Injection, but in general the Decision Tree is better. First of all, the most important features are

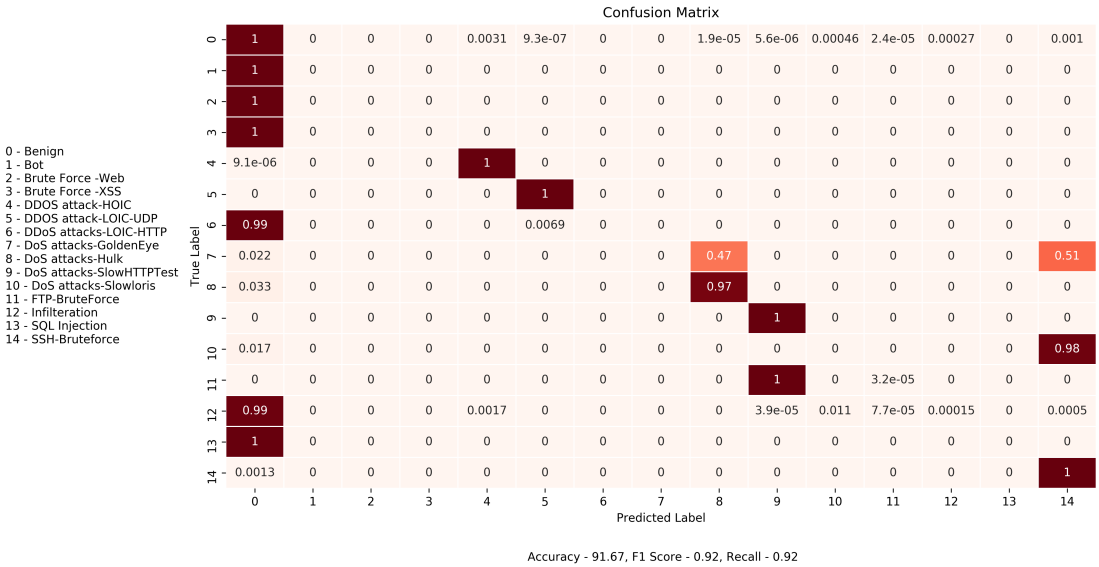


Figure 3.6: Confusion matrix for AdaBoost with 400 stumps (very small Decision Trees).

extracted. Then the most important features are selected by 5-fold cross validation. The cross validation is run for the training data with all features and for the training data with the most important features. The results of both cross validations are compared by t-test whether they significantly differ. The differences in cross validation results are not significant. For all intrusions only the most important features can be used. The list of the most important features and the features, that are selected for the final training, are in Attachment B.1 in the column for the Decision Tree. The same most important features may be used for creating rules, because the Decision Tree can be considered as a set of rules.

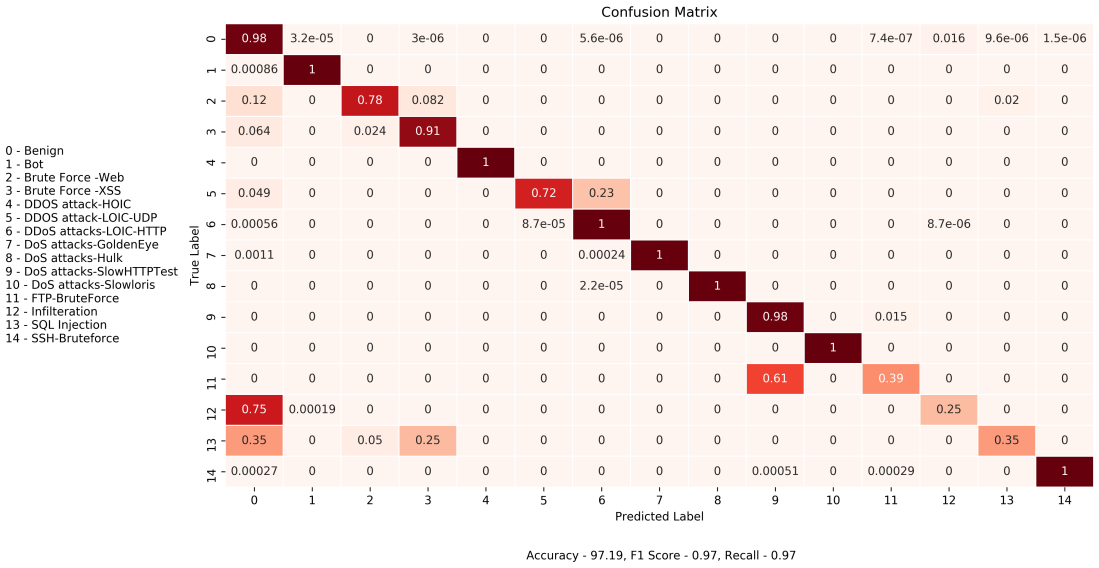


Figure 3.7: Confusion matrix for multiclass Decision Tree (One-vs-All). The Decision Trees are trained on the most important features only.

4. Rules Creation

This chapter discusses creating simple and short rules for each intrusion in the dataset. Section 4.1 starts with describing options of library for creating rules and continues with the rules training and presenting the final rules. The rules training is discussed for each intrusion separately. The last section is Section 4.2. It compares results of the final rules with the Decision Tree results. Let us now dive into the rules training.

4.1 Training

As mentioned before, rules can be created in many ways. In this work, they are created by method that is proposed by Jerome H. Friedman and Bogdan E. Popescu[28][29] and implemented in Python library called Skope-Rules[30]. Rules are extracted from an ensemble of Decision Trees. The Decision Trees perform very well on the intrusion detection problem, so creating rules should not be very difficult, but simple and short rules may not exist or may not perform well. Also it would be great to use only the most important features to not have too complex rules. The feature importance is referred in the previous chapter. The Decision Tree provides feature importance too. Since the Decision Tree can be seen as a set of rules, its feature importance should be the most appropriate. However, rules might need more features than the Decision Tree or might consider different features as important.

Library Skope-Rules provides many parameters how to tune creating rules. Parameters, that are used here, are *n_estimators*, *max_depth*, *precision_min* and *recall_min*. First, the parameter *n_estimators* defines how many estimators (Decision Trees) should be used in the ensemble. The parameter *max_depth* defines the maximum depth of the Decision Tree. This parameter can be used for regulating the maximum length of rules. The last parameters *precision_min* and *recall_min* are used in stage of selecting final rules. They define what minimal performance rules have to have to be selected. The objective is to have only few rules with low *max_depth* and high *precision_min* and *recall_min*.

The rules are created in the same way for each class. The rules should be short and the set of rules should be small. That is why each training starts with the parameter *max_depth* set to 2 and the parameter *n_estimators* set to 30. A higher number of estimators may provide more various rules. The parameters *precision_min* and *recall_min* are tuned for each value of the parameter *max_depth*. If the parameters *precision_min* and *recall_min* are high, then less rules are created. There are usually rules with low Precision and Recall and setting the parameters *precision_min* and *recall_min* to higher value than 0.5 does not create any rules. The parameter *max_depth* is tested for values from the interval $\langle 2, 10 \rangle$. The rules longer than 10 are considered as too long. The training is very sensitive to setting the parameters. A small change in the parameter *max_depth* or in the parameters *precision_min* and *recall_min* makes big difference in results. Many longer rules usually perform worse than short rules. At the end of the training, the values of the rules are returned back to their original values without standardization. The original values may provide more information to human experts than the values

after standardization.

The features for the training are selected based on the Decision Tree. The Decision Tree provides the feature importance for each class (see Attachment B.1). The training uses the most important features, that are provided by the Decision Tree. However, sometimes the rules need more features for the training than the Decision Tree. The features, that are provided by the Decision Tree and selected for the training, are listed in Attachment C.1. More features means longer or more rules, so it is not appropriate to use too many features. If the rules do not perform well, the number of features is increased by 1. The increasing continues until it achieves similar results as the Decision Tree or it contains too many features. On the other hand, in most cases the final rules use less features than the training. The features of the final rules are presented in Attachment C.1 and the performance of the rules is presented in Attachment C.17. All final rules are listed in Attachment C.2. Let us go through each class.

Class Benign represents usual activity without any intrusion. It is not very important for rules, because rules separate only specific intrusion from the other intrusions and from usual activity. If it is not any intrusion, it should do nothing. It means if any rule is not met, it is usual activity (Benign class). On the other hand, it can be seen as if it is not usual activity, it is probably an intrusion. However, classifying Benign class may be more difficult than classifying only specific intrusion. The training of rules uses 25 the most important features, that are extracted from the Decision Tree. Experiments demonstrate that shorter rules have better generalization than longer rules. The parameter *max_depth* provides best results with value 3. Higher values provide the same number of rules but longer rules. However, the rules have lower recall and precision. The set of rules is tested on the test set. The final set of rules has only 2 rules and uses only 3 features. It is presented in Table 4.1. The results in Attachment C.17 show that separating Benign class is not easy for the rules. On the one hand, only 4.7% of intrusions is classified as Benign class, so it is not bad result for intrusion detection. On the other hand, 25% of Benign class is classified as an intrusion, so it may not be appropriate to have too many false negative and to be notified about an intrusion every fourth Benign sample. Classifying only one specific intrusion should be easier and rules should be more accurate. Let us have a look on specific intrusions.

Benign	1.	*Init Fwd Win Byts* ≤ 25742.5261 $\wedge$ *Dst Port* ≤ 18521.5103 $\wedge$ *Subflow Fwd Byts* > 66699.3250
	2.	*Dst Port* > 19464.6142 $\wedge$ *Subflow Fwd Byts* ≤ 66699.3250
Params: *n_estimators* $= 30$, *max_depth* $= 3$, *precision_min* $= 0.3$ and *recall_min* $= 0.1$ *Performance*: *Recall* $= 0.75$, *F1 Score* $= 0.85$		

Table 4.1: Set of rules for class Benign.

The first intrusion is class Bot. The training of rules uses only 2 the most important features. This number of features should not need very long rules. Rectangular area in 2 dimensional space can be defined by 4 points only, so the

appropriate maximum length of rules should be 4. The final set of rules has only 2 rules with length of 3 (see Table 4.2). The rules use only conjunction, but with using disjunction the set of rules can be transformed into one rule. The area, that is defined by the feature *Dst Port*, is same in both rules. Only area, that is defined by the feature *Bwd Pkt Len Mean*, differs. The class Bot can be classified with only one simple rule and at the same time classification can achieve Recall 0.97 and F1 Score 0.99.

Bot	1. $Bwd\ Pkt\ Len\ Mean \leq 85.4608$ $\wedge\ Dst\ Port \leq 18528.3875$ $\wedge\ Dst\ Port > 18527.4706$ 2. $Bwd\ Pkt\ Len\ Mean > 85.4842$ $\wedge\ Dst\ Port \leq 18528.3875$ $\wedge\ Dst\ Port > 18527.4706$
Params: $n\_estimators = 30$, $max\_depth = 3$, $precision\_min = 0.3$ and $recall\_min = 0.1$ *Performance*: *Recall* $= 0.97$, *F1 Score* $= 0.99$	

Table 4.2: Set of rules for class Bot.

Unfortunately, not every class can be distinguished as the previous one. Class Brute Force -Web is not very good separable with a set of rules even with the higher number of longer rules. The rules are trained on 9 the most important features. In comparison with the previous class the parameter *max_depth* has to be set to 7 to get reasonable results. The higher values of *max_depth* have worse results. Moreover, the objective is to create simple and short rules. Longer rules can be extracted directly from the Decision Tree and it does not need a special method for creating rules. Class Brute Force -Web does not contain many samples and it causes a problem in the training. Unfortunately, Skope-Rules does not allow to specify weights of classes, so it has to be solved in a different way. It can be solved in two ways – up-sample a minority class or down-sample a majority class. Here the unbalanced classes are solved by up-sampling the minority class. However, creating rules for this class is not very successful. It creates 5 rules and each has minimum length of 5 and maximum length of 6. Because the rules are too long, they are attached in Attachment C.8. Even with such rules it achieves on the test set Recall 0.57 and F1 Score 0.55 only. On the other hand, this intrusion may not be easy to detect.

Similar class is Brute Force -XSS. Both classes represent web attacks. Class Brute Force -XSS has the same problem. It does not contain many samples. It has approximately the same number of samples like the previous class. The unbalanced classes are solved by up-sampling the minority class. The training uses 20 the most important features and the parameter *max_depth* is set to 10. Shorter rules do not perform well. The final set of rules is more successful than the previous one, but it contains longer rules. The training creates 5 rules, where each rule has minimum length of 4 and maximum length of 9. The rules are attached in Attachment C.9. The rules are complex, but on the test set they achieve Recall 0.83 and F1 Score 0.86.

The rules for classes representing DDoS intrusions are much better. They are more accurate, shorter and more simple. The first class is DDOS attack-HOIC.

DDOS attack-HOIC	1.	*Fwd Pkts/s* $\leq$ 214905.9764 $\wedge$ *Init Fwd Win Byts* $>$ 28852.3201 $\wedge$ *Dst Port* $\leq$ 14678.0294

Params: *n_estimators* = 30, *max_depth* = 3, *precision_min* = 0.3 and *recall_min* = 0.1
Performance: *Recall* = 1.0, *F1 Score* = 0.99

Table 4.3: Set of rules for class DDOS attack-HOIC.

For the training it requires only 5 the most important features. The training creates only one rule with length of 3 and with 3 different features (see Table 4.3). On the test set it gives almost 1.0 for both Recall and F1 Score. The second class is DDOS attack-LOIC-UDP. It uses only 3 the most important features. However, it suffers from unbalanced classes. It needs a small up-sampling of the minority class. The class does not contain many samples and the method for creating rules is not able to create good rules. With up-sampling it creates one rule with length of 3 and with 2 different features. The rule is presented in Table 4.4. This rule is a little bit worse, but still it achieves Recall 1.0. On the other hand, F1 Score is poor and achieves 0.07 only. The last DDoS class is DDoS attacks-LOIC-HTTP. The training uses 5 the most important features. However, the training is unable to find rules with length of 3 and the parameter *max_depth* has to be increased to 10 and the parameter *n_estimators* has to be increased to 40. Also the selection of rules has to be less strict. The parameter *precision_min* is set to 0.2. Only after that the training creates 2 rules with minimum length of 4 and maximum length of 7 (see Attachment C.12). Recall of the rules is a little bit lower, but on the other hand F1 Score is closed to 1.0.

DDOS attack-LOIC-UDP	1.	*Fwd IAT Max* $\leq$ 737245238.1083 $\wedge$ *Fwd IAT Max* $>$ 737241580.5443 $\wedge$ *Fwd IAT Std* $\leq$ 378020739.7917

Params: *n_estimators* = 30, *max_depth* = 3, *precision_min* = 0.3 and *recall_min* = 0.1
Performance: *Recall* = 0.97, *F1 Score* = 0.07

Table 4.4: Set of rules for class DDOS attack-LOIC-UDP.

Next type of intrusion is DoS. The sets of rules for classes representing DoS intrusions perform also well, but they need more rules. Parameters for the training are almost the same for every DoS class. The first class is DoS attacks-GoldenEye. The training uses 10 the most important features and creates 4 rules with length of 5 and with only 6 features (see Attachment C.14). The rules achieve almost 1.0 in both accuracy measures. The second class is DoS attacks-Slowloris. During the training it reduces 23 passed features on only 9 features. The training creates 3 rules with length of 3 and 1 rule with length of 4 (see Attachment C.13). The performance of rules is similar. The third class is DoS attacks-Hulk. In comparison with previous DoS intrusions, this class may be easier to separate. The training creates only 1 rule with length of 3 and with 3 different features. The rule is presented in Table 4.5. From 10 passed the most important features it uses only 3. The rule achieves very similar results as the rules for the previous DoS classes. Last DoS class is DoS attacks-SlowHTTPTest. Here the training

uses 8 the most important features. It creates 2 rules with length of 3 and with 3 different features (see Table 4.6). However, F1 Score is a little bit lower, but still Recall is almost 1.0.

DoS attacks-Hulk	1.	*Fwd IAT Min* $\leq$ 835816163.0157 $\wedge$ *Fwd Seg Size Min* $>$ 35.7566 $\wedge$ *Tot Bwd Pkts* $\leq$ 170.8652

Params: *n_estimators* = 40, *max_depth* = 3, *precision_min* = 0.3 and *recall_min* = 0.1
Performance: *Recall* = 0.97, *F1 Score* = 0.98

Table 4.5: Set of rules for class DoS attacks-Hulk.

Classes representing Brute Force intrusion are easy to detect too. They do not need too long and too many rules. Setting the maximal length to 3 is enough. Class FTP-BruteForce uses only 3 features for the training and the result is 2 rules with length of 3 and 1 rule with length of 2. The rules are attached in Attachment C.4. The rules detect the FTP-BruteForce intrusion very well. They achieve Recall 1.0 and F1 Score 0.73. The last class of the Brute Force intrusion is SSH-BruteForce. The rules need more features for the training. The training creates 2 rules with length of 3 and with 4 different features. The rules are presented in Table 4.7. Their performance is better in case of F1 Score, but Recall is 0.97. On the other hand both classes are classified very well.

The worst intrusion for rules is Infiltration. Even the Decision Tree is not able to fully detect the intrusion and needs many features. In case of creating rules, it is more difficult. The more features it has the more unfeasible it is. The Decision Tree uses 63 features. However, the method for creating rules is not able to finish the training with 63 features. Another problem is the length of rules. With using 63 features rules may have length of 63 and more. It would be too long. The number of features is reduced to finish the training of rules. If the training has only 24 features, it is able to reach the end. However, it is unable to provide any rules. The parameter *max_depth* is increased to 10, but it is unable to create rules even with length of 10. There may be some rules for Infiltration, but the rules may be too long or they may need more features than 24. Longer rules with more features are almost same like the Decision Tree and does not give any advantage in comparison with the Decision Tree.

DoS attacks-SlowHTTP-Tests	1.	*Bwd Pkts/s* $>$ 91987.0181 $\wedge$ *Flow Pkts/s* $\leq$ 783666.1556 $\wedge$ *Fwd Seg Size Min* $>$ 54.1508
	2.	*Flow Pkts/s* $\leq$ 783666.1556 $\wedge$ *Flow Pkts/s* $>$ 533773.4194 $\wedge$ *Fwd Seg Size Min* $>$ 54.1508

Params: *n_estimators* = 40, *max_depth* = 3, *precision_min* = 0.3 and *recall_min* = 0.1
Performance: *Recall* = 0.99, *F1 Score* = 0.68

Table 4.6: Set of rules for class DoS attacks-SlowHTTPTests.

Similar difficult intrusion for detecting is SQL Injection. Compared to Infiltration, SQL Injection does not have too many the most important features and

it is possible to find some rules. The training uses 10 the most important features. It does not need too long rules. The results are achieved with the maximal length set to 5. It is another class that suffers from unbalanced classes. However, the up-sampling works well. The training creates 5 rules with 8 different features (see Attachment C.7). The performance of the rules is similar to the performance of the rules for web intrusions such as Brute Force -Web. F1 Score is very poor and Recall is only 0.55.

The rules are also tested in the multiclass classification. The previous performance results are only for the binary classification and do not give a wider view on rule-based intrusion detection. Each rule is only binary classifier that provides $\{0, 1\}$ values. There is no probability distribution, so only one rule can be selected for classification one sample. During the training rules are selected based on their Precision, Recall and Out-of-Bag error (OOB). These values can be used for selecting the appropriate rule for a sample classification. The rules can be sorted by one of the performance measure. After sorting the rules, the most appropriate rule is the first satisfying rule. The rules are sorted by each performance measure, but the best order of the rules is the order given by Precision. The classification is tested in two ways. The first way is to use all rules. The second way is to use only rules for intrusions and remove the rules for Benign class. However, in both cases there can be a problem that some rules may not satisfy any rule. In the first case, it can be solved by classifying the unrecognized samples as Infiltration. Infiltration is the only class that does not have any rule, so the unrecognized samples may be Infiltration. In the second case, it is more simple. The unrecognized samples are classified as Benign class, because the rules for Benign class are removed. The best performing classification is with using only rules for intrusions. The rules for Benign class are not very good, so it is better to classify unrecognized samples as Benign class than as Infiltration class. The result is presented in Figure 4.1.

Some final rules are only generalization of another rule or more rules can be put together into one rule. In case of binary classification, where order of the rules does not matter, the rules can be simplified. However, in case of multiclass classification it can not be done, because the order of the rules would be lost and the order is very important. Also the rules may be more clear in the original form and human experts can decide which rule to use or how to modify it for their system. Let us now compare the results of the rules and the results of the Decision Tree, that has the best results among tested models.

SSH-BruteForce	1.	*Bwd Pkts/s* > 90301.7439 $\wedge$ *Fwd Header Len* > 12813.2019 $\wedge$ *Dst Port* ≤ 14663.8164
	2.	*Bwd Pkts/s* > 110391.2570 $\wedge$ *Fwd Header Len* ≤ 12813.2019 $\wedge$ *Bwd Header Len* > 3399.2300
Params: *n_estimators* $= 30$, *max_depth* $= 5$, *precision_min* $= 0.3$ and *recall_min* $= 0.1$ *Performance*: *Recall* $= 0.97$, *F1 Score* $= 0.98$		

Table 4.7: Set of rules for class SSH-BruteForce.

4.2 Comparison

This section discusses comparison of the Decision Tree and the set of rules. First of all, the Decision Tree is a more complex model than the set of rules, but the set of rules can be created from the Decision Tree, so the set of rules may reflect some Decision Tree. However, the set of rules should be small and the rules should be short and simple otherwise the Decision Tree can be used instead of rules. The rules for intrusions are created in this manner. The Decision Tree serves to provide only the best results and the set of rules tries to achieve the same results, but it tries to stay as simple as possible. Let us now go through each intrusion and compare both results. The comparison of both results is presented in Attachment C.17.

Class Benign is not fully classified in both cases, but the Decision Tree performs better. The rules have smaller Recall and F1 Score, but for only 2 rules it is not big trade-off between performance and simplicity. For class Bot both classifiers achieve similar good results. Only small difference is in Recall, where it differs by 0.03, but this is the only price for getting only 2 short rules. Next class Brute Force -Web is not easy for both classifiers. The rules are again worse, but both classifiers would not detect all Brute Force -Web intrusions. Similar intrusion Brute Force -XSS has different results. The Decision Tree is better in Recall but has very poor F1 Score. Compared to that, the rules have smaller Recall than the Decision Tree, but has high F1 Score. However, higher Recall may be more appropriate within the intrusion detection. Classes DDOS attack-HOIC and DDoS attacks-LOIC-HTTP are not very interesting, because the results are almost the same, but class DDOS attack-LOIC-UDP is exceptional for the rules. In this case, the rules are better in Recall and the Decision Tree is better in F1 Score. Only 1 good rule is able to detect almost every DDOS attack-LOIC-UDP. Classes representing DoS intrusions have no big difference in results, but the rules perform a little bit worse in case of Recall. Class FTP-BruteForce is another exception. The rules for this class have better Recall than the Decision Tree and have similar F1 Score like the Decision Tree. The worst case for the rules is class Infiltration. Here the rules fail and only the Decision Tree is able to detect at least something, but even the Decision Tree has poor results. It is able to detect only half of Infiltration intrusions. Similar difficult intrusion is SQL Injection. The Decision Tree is able to detect only half of SQL Injection intrusions, but the rules are able to detect more than half. However, rules have very low F1 Score, but the rules may be more suitable for an intrusion detection system. The last class is SSH-BruteForce, where again is no big difference in results. Only 2 short rules can achieve almost the same result like the Decision Tree.

In multiclass classification, it is more clear that the rules have similar results as the Decision Tree. Both the rules and the Decision Tree achieve Precision and Recall 0.97 (see Figure 4.1 and 3.7). The rules are better only in classifying classes – Benign, DDOS attack-LOIC-UDP, FTP-BruteForce and SQL Injection. In other cases, the rules are worse than the Decision Tree except class DDOS attack-HOIC, where the results are same.

In general, the rules have the same results in case of TN and FP for all classes. F1 Score is the same or better for classes Bot, Brute Force -XSS and DoS attack-SlowHTTPTest. TP and FN is the same or better for classes DDoS attack-

HOIC, DDoS attack-LOIC-UDP, FTP-BruteForce and SQL Injection. However, all rules may be used in the intrusion detection system except few rules. From the comparison above can be seen that trade-off between human readability and performance is not very big in most cases. The rules are able to achieve almost the same results like the Decision Tree, but human experts can easily understand them and adapt their intrusion detection systems.

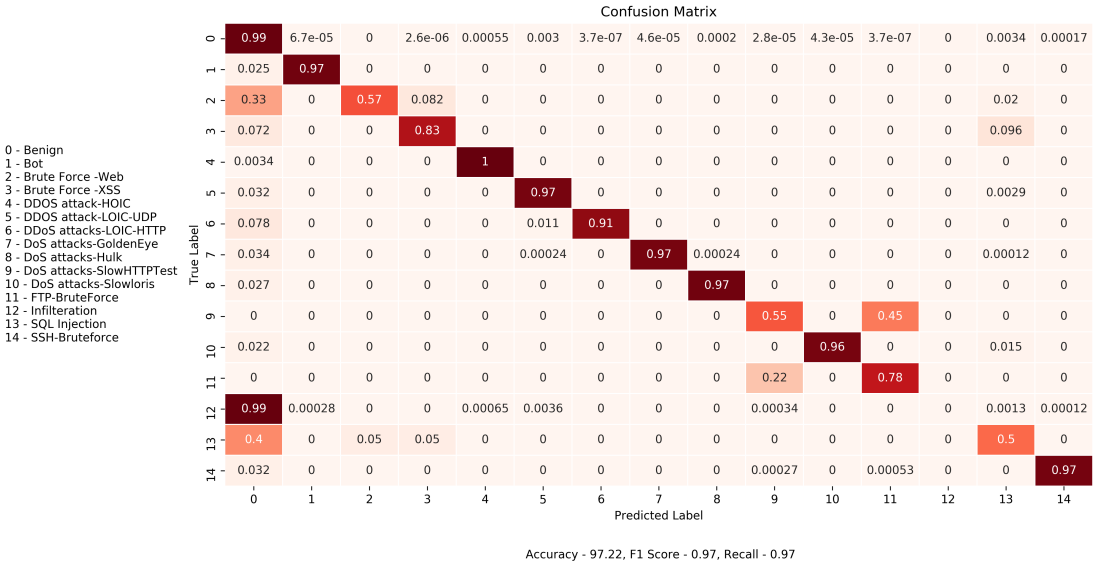


Figure 4.1: Confusion matrix for multiclass rules. The rules are sorted by their performance measure and then the classification is made based on their order. Each sample is classified only by the first satisfying rule. In this case the rules are sorted by their Precision. If any rule is not met, then it is classified as Benign.

Conclusion

This work presents rules for intrusion detection and their comparison with model that achieves the best current results. Models and rules are trained on realistic dataset for intrusion detection created by the Communications Security Establishment and the Canadian Institute for Cybersecurity. The dataset is called CSE-CIC-IDS2018[1][2]. The objective of this work was to create simple and short rules that achieve similar results as the best performing model.

The first practical part explores performance of various models. Linear, non-linear, Decision Tree and ensemble models are tested. Results show that the best model is the Decision Tree. It is able to detect most intrusions. However, some of them are more difficult and it does not perform very well. Compared to results achieved by Qianru Zhou and Dimitrios Pezaros[6], the Decision Tree gives very similar results. These results are used for the comparison with results of the rules.

The second practical part presents the final rules for each intrusion. The rules are created from an ensemble of trees. The method for extracting rules from an ensemble of trees is proposed by Jerome H. Friedman and Bogdan E. Popescu[28][29]. For most intrusions the rules achieve similar or even better results than the Decision Tree and still they are short and simple. However, the rules perform badly for some problematic intrusions such as Infiltration. Infiltration is the only case, where the rules completely fail. On the other hand, it is also difficult for the Decision Tree. Each intrusion is classified by a maximum of five rules, but most intrusions are classified by one or two short rules. The rules use at most 12 features from 80 existing features, but usually the number is lower.

The objective of this work is met and the rules were created for each type of intrusion except one. The intrusion Infiltration is difficult to detect for all models and more difficult for simple rules. The rules are simple and short, so human experts may easily understand them and adapt their defense against intrusions. It may be used in rule-based intrusion systems too. One of the systems that allows to configure custom rules for anomaly detection is IBM QRadar SIEM[1]. However, even without such systems the rules can be deployed as a simple model for intrusion detection. The results were provided to partner organization that uses an intrusion detection system. Unfortunately, because of current epidemic situation it was not possible to compare the results with their intrusion detection system.

Rules in this work are extracted from an ensemble of Decision Trees, but there are also different methods for creating rules. It may be interesting to create rules by different method and compare obtained rules. One of the interesting methods may be Evolutionary Algorithms, namely method Michigan and Pittsburgh. In case of machine learning models, it may be interesting to use an ensemble method Stacking to get better results in difficult intrusions. Each model may be better in detecting different samples and may have different advantages. The Stacking can combine different models together and improve their final results. Another experiment may be to include time of samples. It would require to sort the data

[1]How to create custom rules can be found on page `https://www.ibm.com/support/knowledgecenter/en/SS42VS_7.3.1/com.ibm.qradar.doc/c_qradar_rul_mgt.html`. The main page of IBM QRadar SIEM is on `https://www.ibm.com/products/qradar-siem`.

by their timestamp. Then a model can take in account a history of predictions. For example, if last four predictions are Infiltration with the probability 0.4, then the fifth prediction can increase its probability that it is Infiltration. Simpler experiment may be to select the best classifier for each intrusion and combine them by One-vs-All or One-vs-One method. It may help to get better results without using complicated method.

Bibliography

[1] CSE-CIC-IDS2018 on AWS. https://www.unb.ca/cic/datasets/ids-2018.html, Accessed on 2020-3-18.

[2] A Realistic Cyber Defense Dataset (CSE-CIC-IDS2018). https://registry.opendata.aws/cse-cic-ids2018/, Accessed on 2020-3-18.

[3] Intrusion Detection Evaluation Dataset (CICIDS2017). https://www.unb.ca/cic/datasets/ids-2017.html, Accessed on 2020-3-18.

[4] Arash Habibi Lashkari., Gerard Draper Gil., Mohammad Saiful Islam Mamun., and Ali A. Ghorbani. Characterization of Tor Traffic using Time based Features. In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy - Volume 1: ICISSP*, pages 253–262. INSTICC, SciTePress, 2017.

[5] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP*, pages 108–116. INSTICC, SciTePress, 2018.

[6] Qianru Zhou and Dimitrios Pezaros. Evaluation of Machine Learning Classifiers for Zero-Day Intrusion Detection - An Analysis on CIC-AWS-2018 dataset. *CoRR*, abs/1905.03685, 2019. http://arxiv.org/abs/1905.03685.

[7] V. Kanimozhi and T. P. Jacob. Artificial Intelligence based Network Intrusion Detection with Hyper-Parameter Optimization Tuning on the Realistic Cyber Dataset CSE-CIC-IDS2018 using Cloud Computing. In *2019 International Conference on Communication and Signal Processing (ICCSP)*, pages 33–36, April 2019.

[8] T. Chadza, K. G. Kyriakopoulos, and S. Lambotharan. Contemporary Sequential Network Attacks Prediction using Hidden Markov Model. In *2019 17th International Conference on Privacy, Security and Trust (PST)*, pages 1–3, Aug 2019.

[9] KDD'99 dataset. http://kdd.ics.uci.edu/databases/, Irvine, CA, USA, Accesed on 2020-3-18.

[10] Ugo Fiore, Francesco Palmieri, Aniello Castiglione, and Alfredo De Santis. Network anomaly detection with the restricted Boltzmann machine. *Neurocomputing*, 122:13–23, 2013.

[11] Mostafa A. Salama, Heba F. Eid, Rabie A. Ramadan, Ashraf Darwish, and Aboul Ella Hassanien. Hybrid Intelligent Intrusion Detection Scheme. *Advances in Intelligent and Soft Computing, Springer*, 96:293–303, 2011.

[12] Md. Zahangir Alom, VenkataRamesh Bontupalli, and Tarek M. Taha. Intrusion detection using deep belief networks. In *2015 National Aerospace and Electronics Conference (NAECON)*, pages 339–344, 2015.

[13] Jihyun Kim, Jaehyun Kim, Huong Le Thi Thu, and Howon Kim. Long Short Term Memory Recurrent Neural Network Classifier for Intrusion Detection. *2016 International Conference on Platform Technology and Service (PlatCon)*, pages 1–5, 2016.

[14] Wafaa Anani and Jagath Samarabandu. Comparison of Recurrent Neural Network Algorithms for Intrusion Detection Based on Predicting Packet Sequences. *2018 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–4, 2018.

[15] Intrusion detection evaluation dataset (ISCXIDS2012). `https://www.unb.ca/cic/datasets/ids.html`, Accessed on 2020-3-18.

[16] CICFlowMeter (formerly ISCXFlowMeter). `https://www.unb.ca/cic/research/applications.html#CICFlowMeter`, Accessed on 2020-3-18.

[17] What is a Botnet? `https://usa.kaspersky.com/resource-center/threats/botnet-attacks`, Accessed on 2020-3-20.

[18] What is a Botnet Attack - Definition. `https://www.akamai.com/us/en/resources/what-is-a-botnet.jsp`, Accessed on 2020-3-20.

[19] What is a brute force attack? `https://www.cloudflare.com/learning/bots/brute-force-attack/`, Accessed on 2020-3-20.

[20] What is a denial-of-service attack? `https://www.cloudflare.com/learning/ddos/glossary/denial-of-service/`, Accessed on 2020-3-20.

[21] What is Cross-site Scripting and How Can You Fix it? `https://www.acunetix.com/websitesecurity/cross-site-scripting/`, Accessed on 2020-3-20.

[22] What is SQL Injection (SQLi) and How to Prevent it. `https://www.acunetix.com/websitesecurity/sql-injection/`, Accessed on 2020-3-20.

[23] Thomas M. Mitchell. *Machine learning*. McGraw-Hill, 1997.

[24] Uday Kamath, John Liu, and James Whitaker. *Deep learning for NLP and speech recognition*, chapter Basics of Machine Learning, pages 39–86. Springer, 2019.

[25] Trevor Hastie, Jerome Friedman, and Robert Tisbshirani. *The Elements of statistical learning: data mining, inference, and prediction*, chapter Support Vector Machines and Flexible Discriminants, pages 417–438. Springer, 2017.

[26] Support Vector Machine vs Logistic Regression. `https://medium.com/@george.drakos62/support-vector-machine-vs-logistic-regression-94cc2975433f`, Accessed on 2020-3-27.

[27] Trevor Hastie, Jerome Friedman, and Robert Tisbshirani. *The Elements of statistical learning: data mining, inference, and prediction*, chapter Tree-Based Methods, pages 305–317. Springer, 2017.

[28] Jerome H. Friedman and Bogdan E. Popescu. Predictive learning via rule ensembles. *The Annals of Applied Statistics*, 2(3):916–954, Sep 2008.

[29] Jerome H. Friedman and Bogdan E. Popescu. Gradient Directed Regularization, 2004. `http://statweb.stanford.edu/~jhf/ftp/pathlite.pdf`, Accessed on 2020-3-23.

[30] Skope-Rules. `https://github.com/scikit-learn-contrib/skope-rules`, Accessed on 2020-3-23.

[31] Trevor Hastie, Jerome Friedman, and Robert Tisbshirani. *The Elements of statistical learning: data mining, inference, and prediction*, chapter Unsupervised Learning, pages 485–579. Springer, 2017.

[32] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*, chapter Linear Algebra, pages 45–49. The MIT Press, 2017.

[33] Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research 9*, 2008.

[34] Visualizing Data using t-SNE. `https://wiki.math.uwaterloo.ca/statwiki/index.php?title=visualizing_Data_using_t-SNE`, Accessed on 2020-4-16.

[35] J. D. Hunter. Matplotlib: A 2D graphics environment. *Computing in Science and Engineering*, 9(3):90–95, 2007.

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

List of Figures

List of Tables

A. Attachment - Dataset

Table A.1: Dataset contents

File	Classes
Wednesday-14-02-2018 TrafficForML CICFlowMeter.csv	Benign, FTP-BruteForce, SSH-Bruteforce
Thursday-15-02-2018 TrafficForML CICFlowMeter.csv	Benign, DoS attacks-GoldenEye, DoS attacks-Slowloris
Friday-16-02-2018 TrafficForML CICFlowMeter.csv	Benign, DoS attacks-SlowHTTPTest, DoS attacks-Hulk
Wednesday-21-02-2018 TrafficForML CICFlowMeter.csv	Benign, DDOS attack-HOIC, DDOS attack-LOIC-UDP
Thursday-22-02-2018 TrafficForML CICFlowMeter.csv	Benign, SQL Injection, Brute Force -XSS, Brute Force -Web
Friday-23-02-2018 TrafficForML CICFlowMeter.csv	Benign, SQL Injection, Brute Force -XSS, Brute Force -Web
Thursday-01-03-2018 TrafficForML CICFlowMeter.csv	Benign, Infilteration
Friday-02-03-2018 TrafficForML CICFlowMeter.csv	Benign, Bot

Table A.2: Features Description

Begin of Table A.2	
Feature Name	**Description**
Flow Duration	Flow duration
Tot Fwd Pkts	Total packets in the forward direction
Tot Bwd Pkts	Total packets in the backward direction
TotLen Fwd Pkts	Total size of packet in forward direction
Fwd Pkt Len Max	Maximum size of packet in forward direction
Fwd Pkt Len Min	Minimum size of packet in forward direction
Fwd Pkt Len Mean	Mean size of packet in forward direction
Fwd Pkt Len Std	Standard deviation size of packet in forward direction
Bwd Pkt Len Max	Maximum size of packet in backward direction
Bwd Pkt Len Min	Minimum size of packet in backward direction
Bwd Pkt Len Mean	Mean size of packet in backward direction
Bwd Pkt Len Std	Standard deviation size of packet in backward direction
Flow Byts/s	flow byte rate that is number of packets transferred per second
Flow Pkts/s	flow packets rate that is number of packets transferred per second
Flow IAT Mean	Mean time between two flows
Flow IAT Std	Standard deviation time two flows
Flow IAT Max	Maximum time between two flows
Flow IAT Min	Minimum time between two flows
Fwd IAT Tot	Total time between two packets sent in the forward direction
Fwd IAT Mean	Mean time between two packets sent in the forward direction
Fwd IAT Std	Standard deviation time between two packets sent in the forward direction
Fwd IAT Max	Maximum time between two packets sent in the forward direction
Fwd IAT Min	Minimum time between two packets sent in the forward direction
Bwd IAT Tot	Total time between two packets sent in the backward direction
Bwd IAT Mean	Mean time between two packets sent in the backward direction
Bwd IAT Std	Standard deviation time between two packets sent in the backward direction
Bwd IAT Max	Maximum time between two packets sent in the backward direction

Continuation of Table A.2	
Feature Name	**Description**
Bwd IAT Min	Minimum time between two packets sent in the backward direction
Fwd PSH Flags	Number of times the PSH flag was set in packets travelling in the forward direction (0 for UDP)
Bwd PSH Flags	Number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)
Fwd URG Flags	Number of times the URG flag was set in packets travelling in the forward direction (0 for UDP)
Bwd URG Flags	Number of times the URG flag was set in packets travelling in the backward direction (0 for UDP)
Fwd Header Len	Total bytes used for headers in the forward direction
Bwd Header Len	Total bytes used for headers in the forward direction
Fwd Pkts/s	Number of forward packets per second
Bwd Pkts/s	Number of backward packets per second
Pkt Len Min	Minimum length of a flow
Pkt Len Max	Maximum length of a flow
Pkt Len Mean	Mean length of a flow
Pkt Len Std	Standard deviation length of a flow
Pkt Len Var	Minimum inter-arrival time of packet
FIN Flag Cnt	Number of packets with FIN
SYN Flag Cnt	Number of packets with SYN
RST Flag Cnt	Number of packets with RST
PSH Flag Cnt	Number of packets with PUSH
ACK Flag Cnt	Number of packets with ACK
URG Flag Cnt	Number of packets with URG
CWE Flag Count	Number of packets with CWE
ECE Flag Cnt	Number of packets with ECE
Down/Up Ratio	Download and upload ratio
Pkt Size Avg	Average size of packet
Fwd Seg Size Avg	Average size observed in the forward direction
Bwd Seg Size Avg	Average size observed in the backward direction
Fwd Byts/b Avg	Average number of bytes bulk rate in the forward direction
Fwd Pkts/b Avg	Average number of packets bulk rate in the forward direction
Fwd Blk Rate Avg	Average number of bulk rate in the forward direction

Continuation of Table A.2	
Feature Name	**Description**
Bwd Byts/b Avg	Average number of bytes bulk rate in the backward direction
Bwd Pkts/b Avg	Average number of packets bulk rate in the backward direction
Bwd Blk Rate Avg	Average number of bulk rate in the backward direction
Subflow Fwd Pkts	The average number of packets in a sub flow in the forward direction
Subflow Fwd Byts	The average number of bytes in a sub flow in the forward direction
Subflow Bwd Pkts	The average number of packets in a sub flow in the backward direction
Subflow Bwd Byts	The average number of bytes in a sub flow in the backward direction
Init Fwd Win Byts	Number of bytes sent in initial window in the forward direction
Init Bwd Win Byts	# of bytes sent in initial window in the backward direction
Fwd Act Data Pkts	# of packets with at least 1 byte of TCP data payload in the forward direction
Fwd Seg Size Min	Minimum segment size observed in the forward direction
Active Mean	Mean time a flow was active before becoming idle
Active Std	Standard deviation time a flow was active before becoming idle
Active Max	Maximum time a flow was active before becoming idle
Active Min	Minimum time a flow was active before becoming idle
Idle Mean	Mean time a flow was idle before becoming active
Idle Std	Standard deviation time a flow was idle before becoming active
Idle Max	Maximum time a flow was idle before becoming active
Idle Min	Minimum time a flow was idle before becoming active
End of Table A.2	

Figure A.1: Features Boxplots part 1

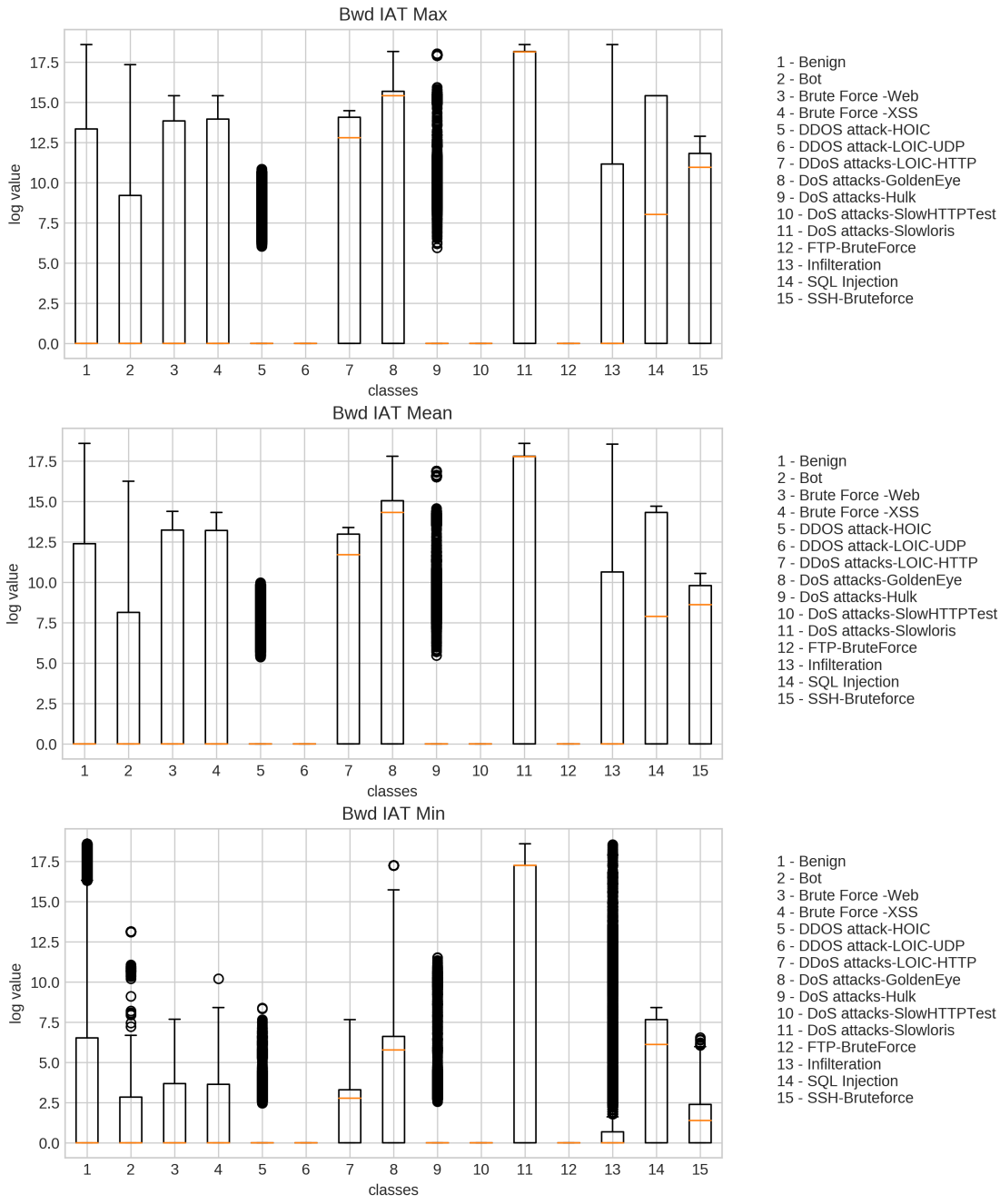


Figure A.1: Features Boxplots part 2

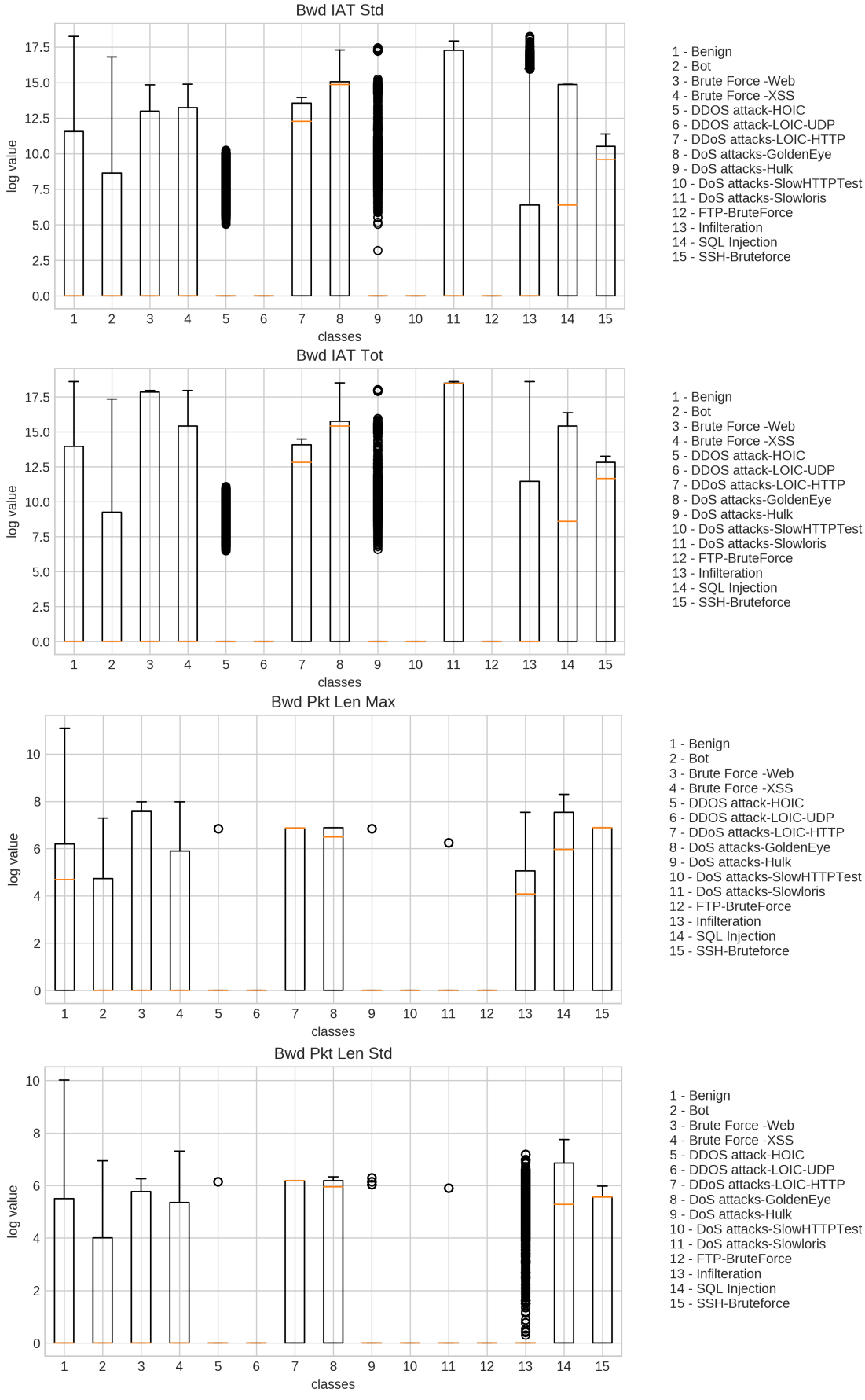


Figure A.1: Features Boxplots part 3

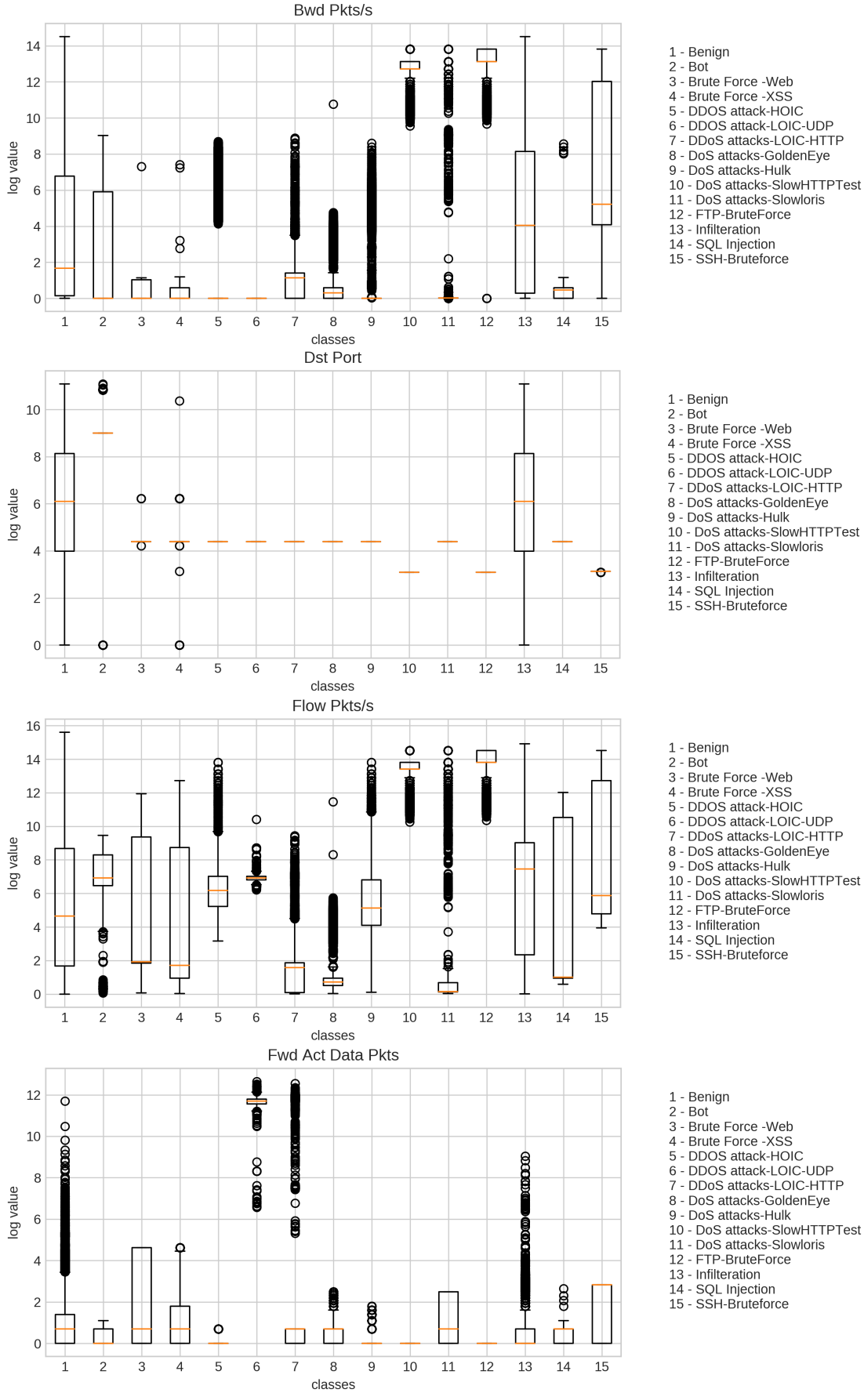


Figure A.1: Features Boxplots part 4

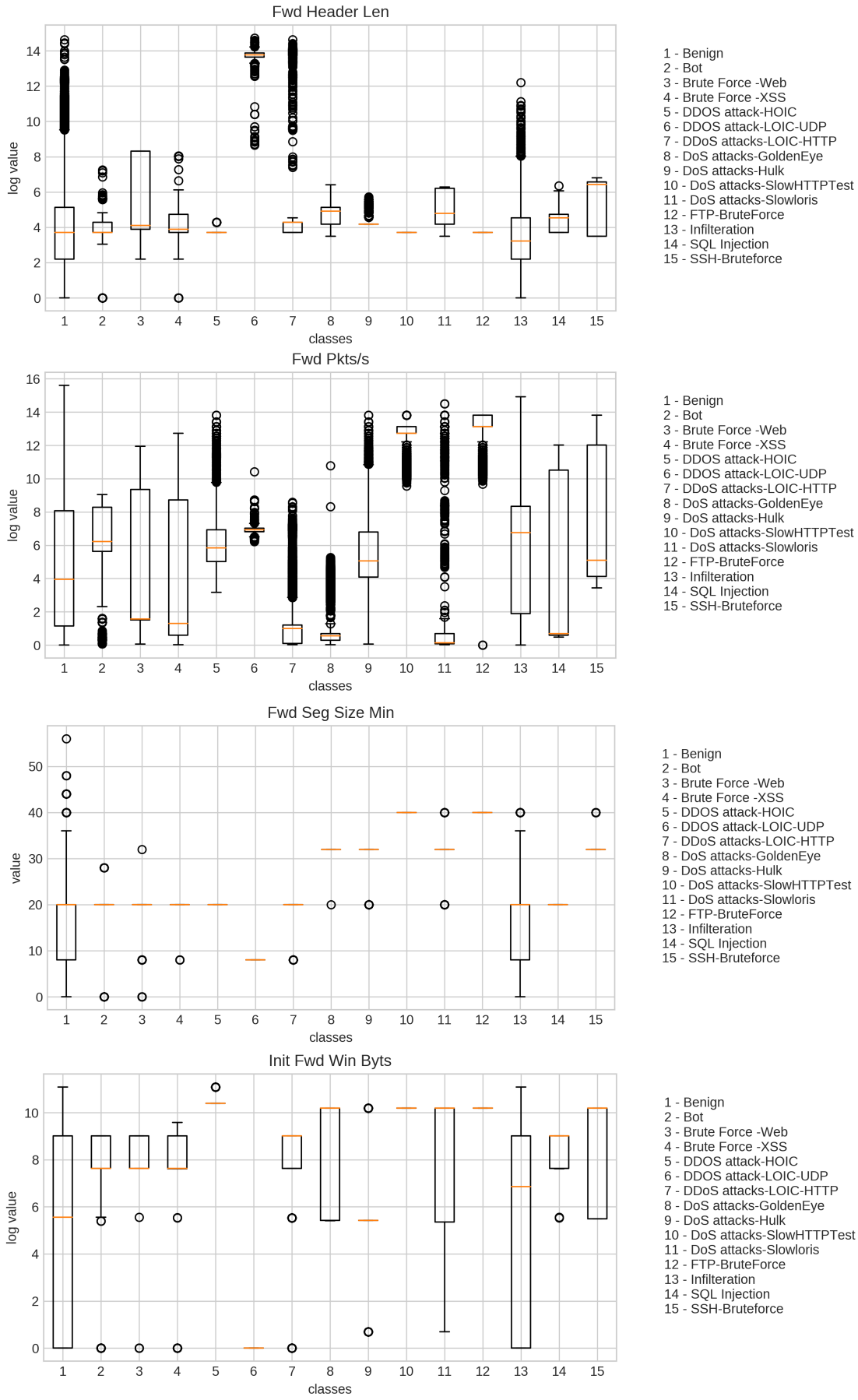
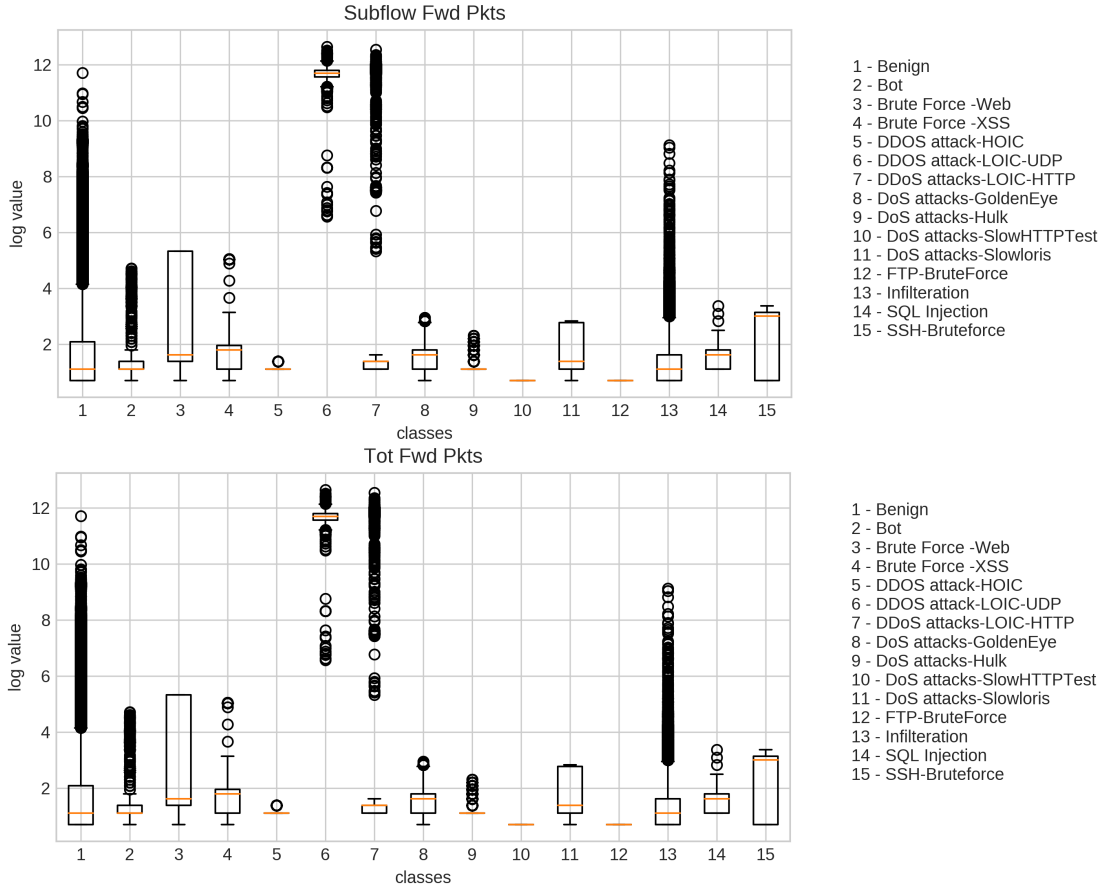


Figure A.1: Features Boxplots part 5



B. Attachment - Model Training

B.1 Feature Importance

Table B.1: Three different feature importance – Logistic Regression, Decision Tree and ANOVA. The features are sorted by their importance.

Begin of Table B.1

Class	Logistic Regression	Decision Tree	ANOVA
Benign	Fwd Seg Size Min, Dst Port, Init Fwd Win Byts, ACK Flag Cnt, Bwd Pkts/s, Init Bwd Win Byts, RST Flag Cnt, ECE Flag Cnt, Fwd Pkt Len Max, Fwd Seg Size Avg, Fwd Pkt Len Mean, Protocol	Dst Port, Subflow Fwd Byts, Init Fwd Win Byts, Fwd Seg Size Min	Fwd Seg Size Min, Bwd Pkts/s, Init Fwd Win Byts, Protocol
Bot	RST Flag Cnt, ECE Flag Cnt, ACK Flag Cnt, Bwd Pkt Len Std, Bwd Pkt Len Max, Init Fwd Win Byts, Protocol, Init Bwd Win Byts	Dst Port, Bwd Pkt Len Mean	RST Flag Cnt, ECE Flag Cnt, Bwd Seg Size Avg, Bwd Pkt Len Mean, Protocol, Bwd Pkt Len Max, Pkt Size Avg
Brute Force - Web	ACK Flag Cnt, Bwd Seg Size Avg, Bwd Pkt Len Mean, Dst Port, Bwd Pkt Len Std, Down/Up Ratio	Init Fwd Win Byts, Dst Port	Bwd Seg Size Avg, Bwd Pkt Len Mean, Pkt Len Mean

Continuation of Table B.1

Class	Logistic Regression	Decision Tree	ANOVA
Brute Force - XSS	ACK Flag Cnt, Dst Port, Fwd Seg Size Avg, Fwd Pkt Len Mean, Protocol, Bwd Pkt Len Min, Init Bwd Win Byts, RST Flag Cnt, ECE Flag Cnt	Bwd Pkts/s, Dst Port, Bwd IAT Max	Pkt Len Min, Fwd Pkt Len Min, Fwd Seg Size Avg, Fwd Pkt Len Mean
DDOS attack-HOIC	Init Fwd Win Byts, ACK Flag Cnt, Dst Port, Init Bwd Win Byts, PSH Flag Cnt	Dst Port, Init Fwd Win Byts	Init Fwd Win Byts
DDOS attack-LOIC-UDP	Fwd Act Data Pkts, Subflow Fwd Pkts, Tot Fwd Pkts, Fwd Header Len	Fwd Act Data Pkts	Fwd Act Data Pkts, Tot Fwd Pkts, Subflow Fwd Pkts, Fwd Header Len
DDoS attacks-LOIC-HTTP	RST Flag Cnt, ECE Flag Cnt, Fwd Pkt Len Mean, Fwd Seg Size Avg, Bwd Pkt Len Std, Fwd Pkt Len Max, Fwd Pkt Len Std, Init Bwd Win Byts, Dst Port	Bwd Pkt Len Std, Flow Duration	RST Flag Cnt, ECE Flag Cnt, Bwd Pkt Len Std
DoS attacks-GoldenEye	Fwd Seg Size Min, Bwd Pkt Len Std, Dst Port, RST Flag Cnt, ECE Flag Cnt, Init Bwd Win Byts, Init Fwd Win Byts	Bwd Pkt Len Std, Flow IAT Min, Flow IAT Mean	Fwd Seg Size Min, Bwd Pkt Len Std
DoS attacks-Hulk	Fwd Seg Size Min, Dst Port, ACK Flag Cnt, PSH Flag Cnt	Tot Bwd Pkts, Fwd Seg Size Min, Fwd IAT Min	Fwd Seg Size Min, ACK Flag Cnt

Continuation of Table B.1			
Class	**Logistic Regression**	**Decision Tree**	**ANOVA**
DoS attacks-SlowHTTPTest	Bwd Pkts/s, Fwd Seg Size Min, PSH Flag Cnt, Init Fwd Win Byts, ACK Flag Cnt	Fwd Seg Size Min	Bwd Pkts/s, Fwd Seg Size Min
DoS attacks-Slowloris	Fwd Seg Size Min, Bwd IAT Mean, Fwd PSH Flags, SYN Flag Cnt	Bwd IAT Max, Fwd Seg Size Min, Pkt Len Mean	Bwd IAT Mean, Bwd IAT Min, Bwd IAT Max
FTP-BruteForce	Bwd Pkts/s, Fwd Seg Size Min, PSH Flag Cnt, ACK Flag Cnt	Fwd Seg Size Min	Bwd Pkts/s
Infiltration[1]	Init Fwd Win Byts, ACK Flag Cnt, Bwd Pkts/s, Fwd Pkt Len Std, ECE Flag Cnt, RST Flag Cnt, PSH Flag Cnt, Fwd Seg Size Min, Fwd PSH Flags, SYN Flag Cnt, Bwd IAT Tot, Down/Up Ratio	Dst Port, Init Fwd Win Byts, Bwd Pkts/s, Fwd IAT Tot, Fwd Seg Size Min, Fwd Pkts/s, Bwd IAT Max, Bwd Pkt Len Max, Fwd IAT Max, Flow Pkts/s, Bwd IAT Min, ACK Flag Cnt	Fwd URG Flags, CWE Flag Count
SQL Injection[2]	ACK Flag Cnt, ECE Flag Cnt, RST Flag Cnt, Init Bwd Win Byts, Dst Port, Bwd Pkt Len Std, Init Fwd Win Byts	Bwd Pkt Len Std, Fwd Pkts/s, Flow Duration, Flow IAT Std, Dst Port	Pkt Len Var, Bwd Pkt Len Std, Pkt Len Std, RST Flag Cnt, ECE Flag Cnt, Bwd Pkt Len Max

[1]Infiltration is difficult to classify and it has many more important features in case of Logistic Regression and Decision Tree, but only the most important are mentioned.

[2]SQL Injection is similar as Infiltration.

Continuation of Table B.1			
Class	**Logistic Regression**	**Decision Tree**	**ANOVA**
SSH-Bruteforce	URG Flag Cnt, Fwd Seg Size Min, Fwd Pkt Len Max, Dst Port, Bwd Pkt Len Max, Init Fwd Win Byts, Pkt Len Max, Init Bwd Win Byts, ECE Flag Cnt, RST Flag Cnt	Bwd Header Len, Dst Port, Fwd Header Len	URG Flag Cnt, Fwd Seg Size Min
End of Table B.1			

B.2 Results

Table B.2: Table compares results of the Logistic Regression and the Decision Tree. Better results are highlighted. Recall is equal to TP in this case. Values TN, FN, TP and FP are normalized. Value $F1$ is F1 Score.

Class	Logistic Regression					Decision Tree				
	TN	FN	TP	FP	$F1$	TN	FN	TP	FP	$F1$
Benign	0.87	0.2	0.8	0.13	0.88	0.95	0.01	0.99	0.05	0.99
Bot	0.82	0.0	1.0	0.18	0.17	1.0	0.0	1.0	0.0	0.99
Brute Force - Web	0.89	0.12	0.88	0.11	0.0	1.0	0.22	0.78	0.0	0.75
Brute Force - XSS	0.76	0.05	0.95	0.24	0.0	1.0	0.02	0.98	0.0	0.02
DDoS attack-HOIC	0.95	0.0	1.0	0.05	0.65	1.0	0.0	1.0	0.0	1.0
DDoS attack-LOIC-UDP	1.0	0.02	0.98	0.0	0.47	1.0	0.28	0.72	0.0	0.83
DDoS attack-LOIC-HTTP	0.82	0.0	1.0	0.18	0.29	1.0	0.0	1.0	0.0	1.0
DoS attack-Golden-Eye	0.89	0.0	1.0	0.11	0.04	1.0	0.0	1.0	0.0	1.0
DoS attack-Hulk	0.81	0.01	0.99	0.19	0.24	1.0	0.0	1.0	0.0	1.0
DoS attack-Slow-HTTP	0.98	0.0	1.0	0.01	0.52	0.99	0.0	1.0	0.01	0.59
DoS attack-Slowloris	0.94	0.0	1.0	0.06	0.02	1.0	0.0	1.0	0.0	1.0
FTP-Brute-Force	0.99	0.0	1.0	0.01	0.65	0.99	0.01	0.99	0.01	0.74
Infiltration	0.68	0.26	0.74	0.32	0.04	0.88	0.54	0.46	0.12	0.07
SQL Injection	0.73	0.35	0.65	0.27	0.0	1.0	0.55	0.45	0.0	0.2
SSH-Brute-Force	1.0	0.0	1.0	0.0	0.88	1.0	0.0	1.0	0.0	1.0

Table B.3: Table compares results of the Logistic Regression and the SVM. Better results are highlighted. Recall is equal to TP in this case. Values TN, FN, TP and FP are normalized. Value $F1$ is F1 Score.

Class	Logistic Regression					SVM				
	TN	FN	TP	FP	$F1$	TN	FN	TP	FP	$F1$
Brute Force - XSS	0.76	0.05	0.95	0.24	0.0	0.93	0.24	0.76	0.07	0.0
DoS attack-Slow-HTTP	0.98	0.0	1.0	0.01	0.52	0.99	0.0	1.0	0.01	0.59
Infiltration	0.68	0.26	0.74	0.32	0.04	0.46	0.31	0.69	0.54	0.03
SQL Injection	0.73	0.35	0.65	0.27	0.0	0.99	0.7	0.3	0.01	0.0

Table B.4: Table compares results of the Decision Tree and the SVM. Better results are highlighted. Recall is equal to TP in this case. Values TN, FN, TP and FP are normalized. Value $F1$ is F1 Score.

Class	Decision Tree					SVM				
	TN	FN	TP	FP	$F1$	TN	FN	TP	FP	$F1$
Brute Force - XSS	1.0	0.02	0.98	0.0	0.02	0.93	0.24	0.76	0.07	0.0
DoS attack-Slow-HTTP	0.99	0.0	1.0	0.01	0.59	0.99	0.0	1.0	0.01	0.59
Infiltration	0.88	0.54	0.46	0.12	0.07	0.46	0.31	0.69	0.54	0.03
SQL Injection	1.0	0.55	0.45	0.0	0.2	0.99	0.7	0.3	0.01	0.0

C. Attachment - Rules

C.1 Features

Table C.1: Table presents the features that are used for the rules training and the features that are used in the final trained rules. The training features are selected based on the Decision Tree feature importance (see Table B.1).

Begin of Table C.1

Class	Training Features	Final Features
Benign	Idle Min, Fwd IAT Tot, Bwd IAT Std, Bwd IAT Mean, PSH Flag Cnt, Active Std, Flow Pkts/s, Fwd Pkt Len Max, Bwd Pkts/s, Bwd IAT Tot, URG Flag Cnt, Bwd IAT Max, Bwd IAT Min, Fwd IAT Mean, Init Bwd Win Byts, Fwd Pkts/s, Fwd Header Len, Flow Duration, Bwd Seg Size Avg, Fwd IAT Std, Bwd Pkt Len Mean, Fwd Seg Size Min, Init Fwd Win Byts, Dst Port, Subflow Fwd Byts	Init Fwd Win Byts, Dst Port, Subflow Fwd Byts
Bot	Bwd Pkt Len Mean, Dst Port	Bwd Pkt Len Mean, Dst Port
Brute Force -Web	Fwd IAT Min, Flow Pkts/s, Down/Up Ratio, Fwd Pkts/s, Bwd IAT Mean, Bwd Pkts/s, Flow IAT Max, Dst Port, Init Fwd Win Byts	Flow Pkts/s, Fwd Pkts/s, Bwd IAT Mean, Flow IAT Max, Dst Port, Init Fwd Win Byts

Continuation of Table C.1		
Class	**Training Features**	**Final Features**
Brute Force -XSS	Fwd IAT Max, Flow IAT Min, Bwd Pkt Len Max, Idle Min, Init Bwd Win Byts, Fwd IAT Min, Fwd Act Data Pkts, Fwd Header Len, Fwd IAT Std, Bwd Pkt Len Std, Bwd IAT Min, Fwd Seg Size Avg, Flow IAT Max, Flow Byts/s, Bwd IAT Tot, Fwd Seg Size Min, Init Fwd Win Byts, Bwd IAT Max, Dst Port, Bwd Pkts/s	Bwd Pkt Len Max, Idle Min, Fwd IAT Std, Bwd Pkt Len Std, Bwd IAT Min, Fwd Seg Size Avg, Flow IAT Max, Bwd IAT Tot, Init Fwd Win Byts, Bwd IAT Max, Dst Port, Bwd Pkts/s
DDOS attack-HOIC	ACK Flag Cnt, Flow IAT Max, Fwd Pkts/s, Init Fwd Win Byts, Dst Port	Fwd Pkts/s, Init Fwd Win Byts, Dst Port
DDOS attack-LOIC-UDP	Idle Min, Fwd IAT Max, Fwd IAT Std	Fwd IAT Max, Fwd IAT Std
DDoS attacks-LOIC-HTTP	Init Fwd Win Byts, Dst Port, Flow IAT Min, Flow Duration, Bwd Pkt Len Std	Init Fwd Win Byts, Dst Port, Flow IAT Min, Flow Duration, Bwd Pkt Len Std
DoS attacks-GoldenEye	Pkt Len Var, TotLen Fwd Pkts, Pkt Len Max, Subflow Bwd Byts, Fwd Seg Size Min, Bwd IAT Std, Init Fwd Win Byts, Flow IAT Mean, Flow IAT Min, Bwd Pkt Len Std	Pkt Len Var, Subflow Bwd Byts, Fwd Seg Size Min, Init Fwd Win Byts, Flow IAT Min, Bwd Pkt Len Std
DoS attacks-Hulk	Fwd Pkt Len Max, Dst Port, Bwd Pkt Len Std, Subflow Bwd Byts, Flow Byts/s, Fwd IAT Min, Fwd Seg Size Min, Tot Bwd Pkts	Fwd IAT Min, Fwd Seg Size Min, Tot Bwd Pkts

Continuation of Table C.1		
Class	**Training Features**	**Final Features**
DoS attacks-SlowHTTPTest	Bwd IAT Std, Bwd IAT Max, Init Fwd Win Byts, Dst Port, Bwd Pkts/s, Flow Pkts/s, Fwd Pkts/s, Fwd Seg Size Min	Bwd Pkts/s, Flow Pkts/s, Fwd Seg Size Min
DoS attacks-Slowloris	Fwd Pkt Len Mean, Flow Byts/s, Active Min, Flow IAT Std, Pkt Len Var, Fwd Header Len, Fwd Act Data Pkts, Flow Duration, Pkt Len Std, Subflow Bwd Pkts, Fwd Pkts/s, Bwd Pkt Len Std, Bwd Pkt Len Max, Pkt Size Avg, Init Fwd Win Byts, Fwd PSH Flags, Idle Min, Fwd IAT Mean, Fwd IAT Min, Dst Port, Pkt Len Mean, Fwd Seg Size Min, Bwd IAT Max	Flow IAT Std, Pkt Len Var, Pkt Len Std, Fwd IAT Mean, Fwd IAT Min, Dst Port, Pkt Len Mean, Fwd Seg Size Min, Bwd IAT Max
FTP-BruteForce	Bwd Pkts/s, Flow Pkts/s, Fwd Seg Size Min	Bwd Pkts/s, Flow Pkts/s, Fwd Seg Size Min
Infiltration	Active Std, Tot Fwd Pkts, Tot Bwd Pkts, Idle Max, Subflow Bwd Pkts, Idle Mean, Idle Std, Subflow Bwd Byts, Active Max, Down/Up Ratio, Active Mean, URG Flag Cnt, Idle Min, Bwd Header Len, Subflow Fwd Byts, Active Min, PSH Flag Cnt, Pkt Len Min, TotLen Fwd Pkts, Fwd Pkt Len Min, Fwd IAT Min, Flow IAT Std, Fwd IAT Std, Fwd Act Data Pkts	–

Continuation of Table C.1		
Class	**Training Features**	**Final Features**
SQL Injection	Pkt Size Avg, Fwd IAT Tot, ECE Flag Cnt, Init Fwd Win Byts, Fwd Seg Size Min, Dst Port, Flow IAT Std, Flow Duration, Fwd Pkts/s, Bwd Pkt Len Std	Pkt Size Avg, ECE Flag Cnt, Init Fwd Win Byts, Dst Port, Flow IAT Std, Flow Duration, Fwd Pkts/s, Bwd Pkt Len Std
SSH-Bruteforce	URG Flag Cnt, Fwd Pkt Len Min, Bwd Pkts/s, Fwd Header Len, Dst Port, Bwd Header Len	Bwd Pkts/s, Fwd Header Len, Dst Port, Bwd Header Len
End of Table C.1		

C.2 Rules Sets

Benign	1. *Init Fwd Win Byts* ≤ 25742.5261 $\wedge$ *Dst Port* ≤ 18521.5103 $\wedge$ *Subflow Fwd Byts* > 66699.3250 2. *Dst Port* > 19464.6142 $\wedge$ *Subflow Fwd Byts* ≤ 66699.3250

Params: *n_estimators* = 30, *max_depth* = 3, *precision_min* = 0.3 and *recall_min* = 0.1
Performance: *Recall* = 0.75, *F1 Score* = 0.85

Table C.2: Set of rules for class Benign.

Bot	1. *Bwd Pkt Len Mean* ≤ 85.4608 $\wedge$ *Dst Port* ≤ 18528.3875 $\wedge$ *Dst Port* > 18527.4706 2. *Bwd Pkt Len Mean* > 85.4842 $\wedge$ *Dst Port* ≤ 18528.3875 $\wedge$ *Dst Port* > 18527.4706

Params: *n_estimators* = 30, *max_depth* = 3, *precision_min* = 0.3 and *recall_min* = 0.1
Performance: *Recall* = 0.97, *F1 Score* = 0.99

Table C.3: Set of rules for class Bot.

FTP-BruteForce	1. *Flow Pkts/s* ≤ 783666.1556 $\wedge$ *Flow Pkts/s* > 671295.0020 $\wedge$ *Fwd Seg Size Min* > 54.1508 2. *Flow Pkts/s* > 783666.1556 $\wedge$ *Fwd Seg Size Min* > 54.1508 3. *Bwd Pkts/s* > 93423.9602 $\wedge$ *Flow Pkts/s* ≤ 671295.0020 $\wedge$ *Fwd Seg Size Min* > 54.1508

Params: *n_estimators* = 30, *max_depth* = 3, *precision_min* = 0.3 and *recall_min* = 0.1
Performance: *Recall* = 1.0, *F1 Score* = 0.73

Table C.4: Set of rules for class FTP-BruteForce.

Infiltration	no simple rules

Params: no parameters
Performance: *Recall* = 0.0, *F1 Score* = 0.0

Table C.5: Set of rules for class Infiltration.

SSH-BruteForce	1. *Bwd Pkts/s* > 90301.7439 $\wedge$ *Fwd Header Len* > 12813.2019 $\wedge$ *Dst Port* ≤ 14663.8164 2. *Bwd Pkts/s* > 110391.2570 $\wedge$ *Fwd Header Len* ≤ 12813.2019 $\wedge$ *Bwd Header Len* > 3399.2300

Params: *n_estimators* = 30, *max_depth* = 5, *precision_min* = 0.3 and *recall_min* = 0.1
Performance: *Recall* = 0.97, *F1 Score* = 0.98

Table C.6: Set of rules for class SSH-BruteForce.

SQL Injection	1. *Init Fwd Win Byts* ≤ 30698.5219 $\wedge$ *Dst Port* ≤ 14765.1416 $\wedge$ *Flow IAT Std* > 378019391.0692 $\wedge$ *Bwd Pkt Len Std* > 643.6374 2. *Pkt Size Avg* ≤ 611.1219 $\wedge$ *Dst Port* ≤ 14765.1416 $\wedge$ *Flow IAT Std* ≤ 378033576.4034 $\wedge$ *Flow IAT Std* > 378018473.6098 $\wedge$ *Bwd Pkt Len Std* > 643.6374 3. *Init Fwd Win Byts* ≤ 12614.7340 $\wedge$ *Dst Port* ≤ 14678.0294 $\wedge$ *Dst Port* > 14675.2785 $\wedge$ *Fwd Pkts/s* > 219712.4913 $\wedge$ *Bwd Pkt Len Std* ≤ 643.6374 4. *ECE Flag Cnt* > 0.5421 $\wedge$ *Flow IAT Std* > 378021155.4142 $\wedge$ *Flow Duration* ≤ 551990567.7851 $\wedge$ *Fwd Pkts/s* ≤ 219718.2201 $\wedge$ *Bwd Pkt Len Std* ≤ 643.6374

Params: *n_estimators* = 30, *max_depth* = 5, *precision_min* = 0.3 and *recall_min* = 0.1
Performance: *Recall* = 0.55, *F1 Score* = 0.0

Table C.7: Set of rules for class SQL Injection.

Brute Force -Web	1. *Fwd Seg Size Avg* > 313.6182 $\wedge$ *Bwd IAT Tot* ≤ 34148496.5648 $\wedge$ *Bwd IAT Max* ≤ 10662392.4841 $\wedge$ *Dst Port* ≤ 30861.6375 $\wedge$ *Bwd Pkts/s* ≤ 90300.9751 $\wedge$ *Idle Min* ≤ 70714799.3529 $\wedge$ *Fwd IAT Std* ≤ 378084753.5283 $\wedge$ *Fwd IAT Std* > 378083434.4581 $\wedge$ *Bwd Pkt Len Std* ≤ 455.4881 2. *Fwd Seg Size Avg* > 313.6030 $\wedge$ *Bwd IAT Tot* > 34148496.5648 $\wedge$ *Bwd IAT Max* ≤ 10662392.4841 $\wedge$ *Dst Port* ≤ 30861.6375 $\wedge$ *Bwd Pkts/s* ≤ 90300.9751 $\wedge$ *Fwd IAT Std* > 378020739.7917 $\wedge$ *Bwd Pkt Len Std* ≤ 455.4881 3. *Bwd IAT Min* ≤ 4059161.2620 $\wedge$ *Bwd IAT Min* > 4058870.6302 $\wedge$ *Fwd Seg Size Avg* ≤ 313.6030 $\wedge$ *Fwd Seg Size Avg* > 126.2039 $\wedge$ *Flow IAT Max* > 737277924.5121 $\wedge$ *Init Fwd Win Byts* ≤ 20784.4100 $\wedge$ *Bwd IAT Max* ≤ 11250202.4434 $\wedge$ *Dst Port* ≤ 14678.0294 $\wedge$ *Bwd Pkt Len Max* > 500.7399 $\wedge$ *Bwd Pkt Len Std* ≤ 371.1062 4. *Fwd Seg Size Avg* ≤ 319.8238 $\wedge$ *Fwd Seg Size Avg* > 313.6182 $\wedge$ *Bwd IAT Max* ≤ 10662392.4841 $\wedge$ *Dst Port* ≤ 30861.6375 $\wedge$ *Bwd Pkts/s* ≤ 90300.9751 $\wedge$ *Fwd IAT Std* > 378020739.7917 $\wedge$ *Bwd Pkt Len Std* ≤ 455.4881 5. *Fwd Seg Size Avg* ≤ 18.2701 $\wedge$ *Init Fwd Win Byts* ≤ 12562.5791 $\wedge$ *Init Fwd Win Byts* > 12547.3142 $\wedge$ *Dst Port* ≤ 14678.0294
Params: *n_estimators* $= 30$, *max_depth* $= 10$, *precision_min* $= 0.3$ and *recall_min* $= 0.1$ *Performance*: *Recall* $= 0.83$, *F1 Score* $= 0.86$	

Table C.8: Set of rules for class Brute Force -Web.

Brute Force -XSS	1. *Flow Pkts/s* $\leq$ 536255.3098 $\wedge$ *Fwd Pkts/s* $>$ 214388.4693 $\wedge$ *Dst Port* $\leq$ 14765.1416 $\wedge$ *Init Fwd Win Byts* $\leq$ 12565.9713 $\wedge$ *Init Fwd Win Byts* $>$ 12565.1233 2. *Flow Pkts/s* $\leq$ 536255.3098 $\wedge$ *Fwd Pkts/s* $\leq$ 226210.9299 $\wedge$ *Fwd Pkts/s* $>$ 214388.4693 $\wedge$ *Dst Port* $\leq$ 14765.1416 $\wedge$ *Init Fwd Win Byts* $\leq$ 12565.9713 $\wedge$ *Init Fwd Win Byts* $>$ 12565.1233 3. *Fwd Pkts/s* $\leq$ 219615.1006 $\wedge$ *Fwd Pkts/s* $>$ 214380.8310 $\wedge$ *Dst Port* $\leq$ 14765.1416 $\wedge$ *Init Fwd Win Byts* $\leq$ 12565.9713 $\wedge$ *Init Fwd Win Byts* $>$ 12565.1233 4. *Bwd IAT Mean* $\leq$ 4529860.4285 $\wedge$ *Bwd IAT Mean* $>$ 4527198.0003 $\wedge$ *Flow IAT Max* $\leq$ 737246449.7924 $\wedge$ *Dst Port* $\leq$ 14690.8670 $\wedge$ *Dst Port* $>$ 14663.8164 $\wedge$ *Init Fwd Win Byts* $>$ 12565.9713 5. *Flow Pkts/s* $>$ 532313.7548 $\wedge$ *Fwd Pkts/s* $\leq$ 219615.1006 $\wedge$ *Fwd Pkts/s* $>$ 214388.4693 $\wedge$ *Dst Port* $\leq$ 14765.1416 $\wedge$ *Init Fwd Win Byts* $\leq$ 12565.9713 $\wedge$ *Init Fwd Win Byts* $>$ 12565.1233
Params: *n_estimators* = 30, *max_depth* = 7, *precision_min* = 0.3 and *recall_min* = 0.1 *Performance*: *Recall* = 0.57, *F1 Score* = 0.55	

Table C.9: Set of rules for class Brute Force -XSS.

DDOS attack-HOIC	1. *Fwd Pkts/s* $\leq$ 214905.9764 $\wedge$ *Init Fwd Win Byts* $>$ 28852.3201 $\wedge$ *Dst Port* $\leq$ 14678.0294
Params: *n_estimators* = 30, *max_depth* = 3, *precision_min* = 0.3 and *recall_min* = 0.1 *Performance*: *Recall* = 1.0, *F1 Score* = 0.99	

Table C.10: Set of rules for class DDOS attack-HOIC.

DDOS attack-LOIC-UDP	1. *Fwd IAT Max* $\leq$ 737245238.1083 $\wedge$ *Fwd IAT Max* $>$ 737241580.5443 $\wedge$ *Fwd IAT Std* $\leq$ 378020739.7917

Params: *n_estimators* = 30, *max_depth* = 3, *precision_min* = 0.3 and *recall_min* = 0.1
Performance: *Recall* = 0.97, *F1 Score* = 0.07

Table C.11: Set of rules for class DDOS attack-LOIC-UDP.

DDoS attacks-LOIC-HTTP	1. *Init Fwd Win Byts* $>$ 15199.5861 $\wedge$ *Flow Duration* $\leq$ 552009298.1353 $\wedge$ *Bwd Pkt Len Std* $\leq$ 432.4779 $\wedge$ *Bwd Pkt Len Std* $>$ 432.4645 2. *Init Fwd Win Byts* $\leq$ 12571.0596 $\wedge$ *Init Fwd Win Byts* $>$ 12306.4684 $\wedge$ *Dst Port* $\leq$ 14678.0294 $\wedge$ *Dst Port* $>$ 14663.8164 $\wedge$ *Flow IAT Min* $>$ 835815284.7113 $\wedge$ *Flow Duration* $\leq$ 552937620.9025 $\wedge$ *Bwd Pkt Len Std* $\leq$ 432.4645

Params: *n_estimators* = 40, *max_depth* = 10, *precision_min* = 0.2 and *recall_min* = 0.1
Performance: *Recall* = 0.91, *F1 Score* = 0.95

Table C.12: Set of rules for class DDoS attacks-LOIC-HTTP.

DoS attacks-Slowloris	1. *Dst Port* $\leq$ 14765.6001 $\wedge$ *Fwd Seg Size Min* $>$ 35.7566 $\wedge$ *Bwd IAT Max* $>$ 29358787.8421 $\wedge$ *Pkt Len Var* $\leq$ 165440.5593 2. *Fwd IAT Mean* $>$ 249110549.4669 $\wedge$ *Fwd Seg Size Min* $>$ 35.7566 $\wedge$ *Bwd IAT Max* $>$ 29358370.0644 $\wedge$ *Pkt Len Std* $\leq$ 172.8473 3. *Pkt Len Mean* $\leq$ 153.3349 $\wedge$ *Fwd Seg Size Min* $>$ 35.7566 $\wedge$ *Bwd IAT Max* $>$ 29358787.8421 $\wedge$ *Pkt Len Var* $>$ 161584.6258 4. *Fwd IAT Mean* $\leq$ 249083966.9954 $\wedge$ *Fwd IAT Min* $>$ 835817956.7616 $\wedge$ *Fwd Seg Size Min* $>$ 54.1508 $\wedge$ *Bwd IAT Max* $\leq$ 29358787.8421 $\wedge$ *Flow IAT Std* $>$ 378017626.7242

Params: *n_estimators* = 40, *max_depth* = 5, *precision_min* = 0.3 and *recall_min* = 0.1
Performance: *Recall* = 0.96, *F1 Score* = 0.96

Table C.13: Set of rules for class DoS attacks-Slowloris.

DoS attacks-GoldenEye	1. *Pkt Len Var* $\leq$ 211890.6134 $\wedge$ *Subflow Bwd Byts* $>$ 244036.7271 $\wedge$ *Fwd Seg Size Min* $>$ 35.7566 $\wedge$ *Init Fwd Win Byts* $\leq$ 26359.9053 $\wedge$ *Bwd Pkt Len Std* $>$ 433.7868 2. *Subflow Bwd Byts* $>$ 244036.7271 $\wedge$ *Fwd Seg Size Min* $>$ 35.7566 $\wedge$ *Init Fwd Win Byts* $\leq$ 26359.9053 $\wedge$ *Bwd Pkt Len Std* $\leq$ 500.8440 $\wedge$ *Bwd Pkt Len Std* $>$ 433.7868 3. *Fwd Seg Size Min* $>$ 35.7566 $\wedge$ *Init Fwd Win Byts* $\leq$ 11596.6518 $\wedge$ *Init Fwd Win Byts* $>$ 11590.2914 $\wedge$ *Flow IAT Min* $>$ 835815745.7487 $\wedge$ *Bwd Pkt Len Std* $\leq$ 433.7868 4. *Subflow Bwd Byts* $\leq$ 244032.6823 $\wedge$ *Fwd Seg Size Min* $>$ 35.7566 $\wedge$ *Flow IAT Min* $\leq$ 835815745.7487 $\wedge$ *Bwd Pkt Len Std* $\leq$ 433.7868 $\wedge$ *Bwd Pkt Len Std* $>$ 311.7684
Params: *n_estimators* = 40, *max_depth* = 5, *precision_min* = 0.3 and *recall_min* = 0.1 *Performance*: *Recall* = 0.97, *F1 Score* = 0.97	

Table C.14: Set of rules for class DoS attacks-GoldenEye.

DoS attacks-Hulk	1. *Fwd IAT Min* $\leq$ 835816163.0157 $\wedge$ *Fwd Seg Size Min* $>$ 35.7566 $\wedge$ *Tot Bwd Pkts* $\leq$ 170.8652
Params: *n_estimators* = 40, *max_depth* = 3, *precision_min* = 0.3 and *recall_min* = 0.1 *Performance*: *Recall* = 0.97, *F1 Score* = 0.98	

Table C.15: Set of rules for class DoS attacks-Hulk.

DoS attacks-SlowHTTP-Tests	1. *Bwd Pkts/s* $>$ 91987.0181 $\wedge$ *Flow Pkts/s* $\leq$ 783666.1556 $\wedge$ *Fwd Seg Size Min* $>$ 54.1508 2. *Flow Pkts/s* $\leq$ 783666.1556 $\wedge$ *Flow Pkts/s* $>$ 533773.4194 $\wedge$ *Fwd Seg Size Min* $>$ 54.1508
Params: *n_estimators* = 40, *max_depth* = 3, *precision_min* = 0.3 and *recall_min* = 0.1 *Performance*: *Recall* = 0.99, *F1 Score* = 0.68	

Table C.16: Set of rules for class DoS attacks-SlowHTTPTests.

C.3 Results

Table C.17: Table compares results of the simple rules and the Decision Tree. Better results are highlighted. Recall is equal to TP in this case. Values TN, FN, TP and FP are normalized. Value $F1$ is F1 Score.

Class	Simple Rules					Decision Tree				
	TN	FN	TP	FP	$F1$	TN	FN	TP	FP	$F1$
Benign	0.95	0.25	0.75	0.05	0.85	0.95	0.01	0.99	0.05	0.99
Bot	1.0	0.03	0.97	0.0	0.99	1.0	0.0	1.0	0.0	0.99
Brute Force - Web	1.0	0.43	0.57	0.0	0.55	1.0	0.22	0.78	0.0	0.75
Brute Force - XSS	1.0	0.17	0.83	0.0	0.86	1.0	0.02	0.98	0.0	0.02
DDoS attack-HOIC	1.0	0.0	1.0	0.0	0.99	1.0	0.0	1.0	0.0	1.0
DDoS attack-LOIC-UDP	1.0	0.03	0.97	0.0	0.07	1.0	0.28	0.72	0.0	0.83
DDoS attack-LOIC-HTTP	1.0	0.09	0.91	0.0	0.95	1.0	0.0	1.0	0.0	1.0
DoS attack-Golden-Eye	1.0	0.04	0.97	0.0	0.97	1.0	0.0	1.0	0.0	1.0
DoS attack-Hulk	1.0	0.03	0.97	0.0	0.98	1.0	0.0	1.0	0.0	1.0
DoS attack-Slow-HTTP	0.99	0.01	0.99	0.01	0.68	0.99	0.0	1.0	0.01	0.59
DoS attack-Slowloris	1.0	0.04	0.96	0.0	0.96	1.0	0.0	1.0	0.0	1.0
FTP-Brute-Force	0.99	0.0	1.0	0.01	0.73	0.99	0.01	0.99	0.01	0.74
Infiltration	1.0	1.0	0.0	0.0	0.0	0.88	0.54	0.46	0.12	0.07
SQL Injection	1.0	0.45	0.55	0.0	0.0	1.0	0.55	0.45	0.0	0.2
SSH-Brute-Force	1.0	0.03	0.97	0.0	0.98	1.0	0.0	1.0	0.0	1.0

D. Attachment - Digital Content

Results:

- folder *statistics* – It contains all computed statistics. The statistics are stored as CSV files. There are statistics for the whole dataset, train set and for each class.
- folder *graphs* – It contains all created graphs.
  - subfolders *box_plots* and *box_plots_log* – They contain created box plots for each feature. The subfolder *box_plots* contains graphs without any feature transformation and the subfolder *box_plots_log* contains graphs with logarithm of features.
  - subfolder *visualization* – It contains the visualized data in 2 dimensional space with using PCA.
  - subfolder *feature_importance* – It contains graphs of various feature importance.
  - subfolder *confusion_matrix* – It contains all confusion matrices. There are confusion matrices for each model, for each class and for multiclass classification.

Python scripts:

- README.md – It contains a description of the scripts.
- script *python/constants.py* – It contains definitions of constants that are used in another scripts.
- script *python/data.py* – It contains an implementation of Dataset class and statements for downloading the dataset.
- script *python/data_analysis.py* – It contains an implementation of Analyzer class and methods for analyzing the dataset.
- script *python/data_transformation.py* – It contains an implementation of Transformer class and methods for the data transformation.
- script *python/data_visualization.py* – It contains methods for the data visualization.
- script *python/graphs.py* – It contains methods for creating various graphs.
- script *python/model_training.py* – It contains methods for computing the feature importance, feature selection and model training.
- script *python/rules.py* – It contains methods for creating rules.
- folder *python/conf* – It contains configurations for the analysis.