Bachelor's thesis

Batchelor's Programme in Computer Science

Machine learning methods for intrusion detection

Kosonen Antti

May 14, 2023

Faculty of Science

University of Helsinki

Contact information

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki,Finland

Email address: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Faculty of Science	Batchelor's Programme in Computer Science

Tekijä — Författare — Author

Kosonen Antti

Työn nimi — Arbetets titel — Title

Machine learning methods for intrusion detection

Ohjaajat — Handledare — Supervisors

Prof. V. Niemi and Dr. M.T. Damir

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Bachelor's thesis	May 14, 2023	23 pages, 3 appendix pages

Tiivistelmä — Referat — Abstract

Intrusions into computer systems cause large costs on society, which is why protecting them is crucial. An intrusion detection system (IDS) distinguishes between malicious and benign events in network data or various types of log files. Traditional IDSs fail to detect many intrusion attempts, especially previously unseen ones. Machine learning is a promising technology for improving detection performance.

This thesis aims to research machine learning methods for intrusion detection, focusing on classification. Experiments on two methods — Naïve Bayes and Random Forest — are conducted using the UWF-ZeekData22-dataset. From the dataset the three most common labels were selected for use in the experimentation The methods are evaluated on statistical metrics.

Experimentation was conducted with two approaches, one with the whole dataset, and the other with a balanced subset. The first approach yielded poor results, which was expected, since the selected models do not work well with heavily imbalanced data. The second approach presented significantly better performance, with Random Forest being the more precise of the two. We conclude that, with more complex models and broader datasets, machine learning methods can be effective tools in intrusion detection.

ACM Computing Classification System (CCS)
Security and privacy → Intrusion detection systems
Computing methodologies → Machine learning

Avainsanat — Nyckelord — Keywords

intrusion detection, machine learning

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Contents

1 Introduction

Information security has widespread societal implications. The World Economic Forum estimates "Widespread cybercrime and cyber insecurity" to be one of the top ten global risks by severity over the period 2023-2033 [5]. In their assessment, cybercrime poses significant risk to privacy, business activity, and even food security and political stability.

An intrusion means gaining access to information in computer systems or damaging system operation in an illegal or unauthorized manner [8]. These can take a wide variety of forms, ranging from attempted break-ins from the outside, to authorized malicious insiders installing malware on systems. Intrusions play a key role in most types of cybercrime, and therefore detecting them early and consistently is pivotal.

Traditional intrusion detection systems have many shortcomings, which can be alleviated with machine learning [8]. In the experimental section of this thesis two machine learning classifiers were trained on a dataset comprising of labeled network sessions to study their suitability for intrusion detection.

Chapter 2 introduces Intrusion detection systems, how they attempt to detect intrusions, and how intrusions are classified. Chapter 3 describes the Machine learning concepts and models used in the experimental section of this thesis, which is then presented in the following two chapters. The dataset, preprocessing, training and metrics are described in Chapter 4, and the results are presented in Chapter 5. Finally, the results are discussed in Chapter 6 and Chapter 7 presents the conclusions of the thesis and suggestions for future work.

2 Intrusion detection systems

An intrusion detection system (IDS) is an application that aims to protect networks and systems by detecting intrusions and generating alerts about them [8]. Some IDSs also include functionality for responding to intrusions. The first IDS was proposed in 1980 in [1], and they have played a key role since then in protecting networks. However, IDSs still have many shortcomings. They still have high false positive rates, burdening security analysts, and they fail to detect many intrusions, especially previously unseen ones. Therefore, IDSs need to be developed to bring down the number of false positives, and to broaden the detection scope. *Machine learning* is a promising technology for advancing towards these goals. It is a type of artificial intelligence technique that can be used to extract useful information from large amounts of data.

2.1 Classification of intrusions

Intrusions into computer networks have existed as long as there have been computer networks. Early intrusions usually involved computer viruses, so conventional incident response mechanisms evolved to mainly combat this threat [7]. They therefore mitigated intrusion risk with two assumptions:

1. response should happen after compromise, and
2. compromise is a result of a fixable flaw.

These kind of methods fail to sufficiently address newer, more focused and manually operated, intrusions that are intent on the compromise of data for economic or military advancement, which do not necessarily employ fixable flaws, and for which response after compromise is too late. To address these shortcomings, various frameworks have been put forward to help analyze intrusions from a threat focused approach. An early example of such a framework is the Cyber Kill Chain by Lockheed Martin, which was based on a United States Department of Defense framework for military strategy.

The MITRE Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK) framework is a newer behavioral model of known adversary behavior [12]. It reflects various

phases of the lifecycle of an intrusion, broken down into tactics, techniques, and sub-techniques. The focus of the framework is on how external adversaries compromise and operate within computer information networks. The first version of the framework was released in 2013, and the current version is version 12, released on October 25, 2022.

The framework was created out of a need for a systematic way to categorize adversary behavior in structured adversary emulation exercises, with the goal of improving post-compromise detection of threats [12]. The metric for success was how well known adversary behaviour was being detected and documented with telemetry sensing and behavioral analytics, and for this categorization of real-world adversary behaviour was deemed useful. In addition to detection and documentation, the framework was also used in emulating those adversaries.

Tactics represent the short-term tactical objectives that are attempted as part of an intrusion, such as discovery of information, persistence of access, lateral movement, or execution of code on target systems [12]. The current version of the ATT&CK framework has 14 tactics. *Techniques* represent the means by which tactics can be attained, and *sub-techniques* further break down techniques into more specific descriptions. A tactic can have one or more techniques linked to it, and any technique can have one or more sub-techniques. Any technique is linked to at least one tactic, but can be linked to multiple, and a sub-technique can be directly linked to one or more tactics. The current ATT&CK framework version has 193 Techniques and 401 Sub-techniques.

Data sources for intrusion detection

In order to detect intrusions, IDSs monitor various data objects - intrusion detection is fundamentally a data-analysis problem [8]. IDSs can be classified by the data source they utilize, which is discussed in this section.

Network-based IDSs attempt to detect intrusions in network traffic [8]. Typically the analysis is done on one or more of the following:

1. *Packets*, which are the basic unit in network communication. Packets include a header and a payload, and either or both of these can be parsed and analyzed in packet-based detection.

2. *Network flow*, which is the set of packets within a time window. Network flow can form the basis of flow-based detection, and it can be grouped analyzed in various

ways.

3. *Sessions*, which is the sequence of communication between two hosts, defined in this context as a 5-tuple (client IP, client port, server IP, server port, protocol). Session-based detection analyzes features of this sequence, or its statistics.

Network-based IDSs are deployed in key network nodes, with the aim of capturing all traffic in a network [8]. However they can only see the traffic traversing through the nodes that the system is monitoring. Network-based IDSs have high detection efficiency: they can detect intrusion attempts in real time, and trace their origins based on network addresses and timestamps.

Host-based IDSs analyze *logs* from operating systems and applications [8]. Logs are usually text files containing detailed information about system performance, user actions and application activity. Host-based IDSs rely on text analysis to decipher log features. Host-based IDSs need to be deployed on every *host* - a computer running an operating system, such as a server or a laptop - which makes deployment difficult and consumes host resources. Network traffic can only be analyzed for the network interfaces of the host. The amount of log data can also be large, requiring considerable amounts of resources for analysis. On the other hand, logs record the complete intrusion on a host, so log analysis results are easily interpretable.

2.2 Methods of intrusion detection

In addition to data sources, IDSs can also be classified by the detection methods they employ. This section discusses the different methods that are traditionally used.

Signature based detection involves the creation of a database of malicious behaviour signatures, which observed behaviour is then compared against [8]. This approach presents many challenges. Signatures need to be generated for all misuse types, and any malicious behaviour without a matching signature is missed. The missed alarm rate is therefore high, and unknown intrusions are left completely unseen. The signature database inevitably grows to a huge size, and it needs to be maintained and updated continually. Creation of signatures requires detailed domain knowledge. On the other, hand false positives are rare, and detections can easily be supplemented with context and an explanation of the intrusion type and impact from the database.

Anomaly detection requires the recording of a normal behaviour profile, and detections are based on behaviour deviating from this baseline by some selected degree [8]. This method requires only limited domain knowledge and can recognize unknown intrusions, but the false alarm rate is high, and the reason for reported abnormalities is not easily investigated. The effectiveness of this approach rests heavily on how well defined and truthful the baseline profile is. If the dataset used as normal behaviour includes behaviour that is in fact malicious, then the baseline profile ends up including the malicious behaviour.

Hybrid methods use signature based detection and anomaly detection in conjunction [8]. For example, some signature based detection tools generate masses of alerts, most of which are not associated with malicious behaviour. These alert logs can then be used as input in an anomaly detection algorithm or a clustering method, to attempt to separate true positives from the mass of logs, or to rank them by severity. In this way limited analyst resources can be used more efficiently, since the analysts can concentrate on the most important alerts.

Machine learning, which is discussed in the next chapter, can be based on any of the three methods described here, depending on the machine learning model.

3 Machine learning models for intrusion detection

Machine learning models can be categorized in two ways: on the one hand they can be supervised or unsupervised learning models; and on the other they can be shallow or deep learning models [8]. Supervised models rely on labeled data, and usually the goal of the model is classification, that is to map the input to a label. Since labeled data is expensive to create, supervised learning is often hindered by lack of labeled data. Unsupervised learning on the other hand uses unlabeled data, which it uses to extract feature information. The detection performance of unsupervised methods is usually inferior to supervised methods.

Shallow models are also sometimes called traditional machine learning models, since some of them have been studied for decades and their methodology is mature [8]. Besides detection, some of these models are also used for practical purposes, such as improving detection efficiency.

Like their name suggests, deep learning methods utilize deep networks [8]. They are a relatively new field of technology, since their study has increased significantly only since 2015. Deep learning methods have significant advantages over shallow models for large datasets. Network architecture, hyperparameter selection and optimization strategies are some of the principal fields of study for deep learning models.

Deep learning models have higher complexity than shallow models, which results in longer training, optimizing, and running times [8]. Their complexity also means that they are effectively black boxes, whereas some shallow models have strong interpretability. On the other hand, deep learning models do not require feature engineering since the models can learn features from the data themselves. Deep learning models have stronger fitting ability, but they also have a higher tendency to overfit, when compared with shallow models.

Hybrid and ensemble methods attempt to combine the strengths and overcome the weaknesses of individual classifiers to achieve better results than single classifiers by themselves [8]. In ensemble methods several different classifiers are trained, and they then vote for the final result. In hybrid methods different classifiers are used in stages.

In this chapter we first present the concept of supervised learning [8]. The two models

Random Forest and Naïve Bayes used in the experimental section, and the underlying decision tree model, are presented in more detail. Finally, the concept of deep learning is presented.

3.1 Supervised learning

Supervised learning is done on *labeled* training data, meaning that each data point in the training set is labeled with the correct classification [8]. The machine learning algorithm can then be trained to infer the label from the features [10]. That is, given a training set $(\vec{x_1}, y_1), (\vec{x_2}, y_2)...(\vec{x_n}, y_n)$, where $\vec{x_i}$ are the features of the instance, and y_i is the label. Assuming that each pair is generated by some unknown function $y = f(\vec{x})$, the goal is to construct a function $h(\vec{x})$ that approximates the function f. The function h is drawn from a function class $\mathcal{H}$.

Utility is the internal, subjective value of an outcome [10]. The objective of the selection of the function h is minimizing the *loss function*, which is defined as the amount of utility lost by predicting $h(\vec{x}) = \hat{y}$ when the correct answer is $f(\vec{x}) = y$ [10], more precisely

$$L(\vec{x}, y, \hat{y}) = U(y, \vec{x}) - U(\hat{y}, \vec{x}), \tag{3.1}$$

where

- $U(y, \vec{x})$ is the utility of the result of using the true label y given an input $\vec{x}$, and
- $U(\hat{y}, \vec{x})$ is the utility of the result of using the predicted label $\hat{y}$ given an input $\vec{x}$.

The machine learning model used defines the function class that is searched for the optimal function, and the loss function that is used to evaluate the function candidates [10]. Some commonly used loss functions are:

- absolute-value loss: $L_1(y, \hat{y}) = |y - \hat{y}|$,
- squared-error loss: $L_2(y, \hat{y}) = (y - \hat{y})^2$,
- 0/1 loss: $L_{0/1}(y, \hat{y}) = 0$ if $y = \hat{y}$, otherwise 1

3.2 Naïve Bayes classification

The Naïve Bayes algorithm is a simple probabilistic classification algorithm based on *Bayes' Theorem* [9]. The theorem is a way to estimate the probability of an event based on the combination of prior knowledge and new information [4]. The theorem is stated mathematically as

$$P(A|B) = P(A)\frac{P(B|A)}{P(B)}, \tag{3.2}$$

where

- A is the event whose probability we are estimating
- B is the observed (new) event, which we are using to update our belief about the probability of A
- $P(A|B)$ is the *conditional probability* of A occurring if B has occurred ("A *given* B")
- $P(B|A)$ is the reverse conditional probability: that of B occurring if A has occurred ("B *given* A")
- $P(A)$ and $P(B)$ are the probabilities of A and B occurring.

The theorem can be understood as updating the degree of belief in the probability of an event, given new evidence [4]. The observer has an initial degree of belief in a, the *prior* $P(A)$, and the incorporating the observation of B occurring gives the *posterior* $P(A|B)$. This is reached by multiplying the prior $P(A)$ with the quotient $\frac{P(B|A)}{P(B)}$ representing the support for A given by the observation of B.

In Naïve Bayes classification what is being updated is the belief that some instance, represented by feature vector $\vec{x}$, belongs to class C_i [9]. That is, the class is A in the theorem, and the instance feature vector is B:

$$P(C_k|\vec{x}) = P(C_k)\frac{P(\vec{x}|C_k)}{P(\vec{x})} \tag{3.3}$$

Since the denominator does not depend on the class, only the numerator is of interest. The algorithm also assumes that the features $x_1...x_n$ are mutually independent, which is one reason it is called "naïve". Therefore, the equation can be rewritten as:

$$P(C_k|x_1...x_n) = \frac{1}{Z}P(C_k)\prod_{i=1}^{n}P(x_i|C_k), \tag{3.4}$$

where Z is a constant scaling factor derived from the features of the instance:

$$Z = P(\vec{x}) = \sum_k P(C_k)P(\vec{x}|C_k) \tag{3.5}$$

The classification of an instance is done by calculating the probabilities of each class with Equation 3.4, and selecting the one that gives the largest probability.

3.3 Random Forest classification

The decision tree algorithm uses a set of rules to classify data in a simple, interpretable way [9]. In training, the dataset is repeatedly split into subsets based on some partitioning of a feature. The splitting starts from some input feature node, from which arcs lead to subordinate decision nodes on a different input feature. This process is repeated on each derived subset recursively, and it is completed when the subset at a node has only one class, or when splitting no longer adds value to the predictions.

In the Random Forest classification method a large number of decision trees are constructed, and the output is the class that is selected by most trees [6]. It is therefore an ensemble learning method, since it is a combination of methods. Decision tree classifiers are prone to *overfitting*, especially when they are deep - they end up corresponding too closely the training data, and their prediction ability with new data is low. In a random forest several different decision trees trained on different parts of the same training set are averaged, which greatly boosts performance.

The technique used in training random forest models is called *bootstrap aggregating*, where the algorithm repeatedly selects a random sample from the training set and fits a decision tree to this sample [6]. The random selection is done *with replacement*, meaning that the selection can include an instance more than once.

4 Methods

This chapter describes the dataset and methodology used in this thesis. A comparative study between two shallow supervised classification methods for intrusion detection was conducted. Section 4.1 describes the dataset used, Section 4.2 describes the feature selection, preprocessing and training, and finally Section 4.3 describes the metrics for comparison of results.

4.1 Dataset

The *UWF-ZeekData22-dataset* is a network data repository collected using *Zeek*, an open-source traffic analyzer [3]. The data was collected from the University of Western Florida Cyber Range, where participants can practice attacking and defending computer infrastructure. The participants reported what type Mitre ATT&CK tactics they were executing at each point in time, and the corresponding rows in the Zeek logs were labeled with this information. Those rows which did not include any attacks were labeled with *none* as the attack tactic. Table 4.1 lists the column names (attributes), their explanations and sample data from the dataset.

The counts of the various tactics in the dataset are presented in Table 4.2 [3]. As we can see, they are skewed: the Discovery-, Reconnaissance- and none-classes have several orders of magnitude more samples than the other classes. For this reason only these three classes were selected for the experimentation.

4.2 Experimental setup

In this section we first discuss feature selection and preprocessing of the data, and finally the training methods are described. The *scikit-learn* machine learning library [11] was used in both preprocessing and training.

Column	Explanation	Example data
resp_pkts	Number of packets responder sent	2
service	Identification of application protocol being sent over connection	dns
orig_ip_bytes	Number of IP level bytes originator sent	186
local_resp	Records if the connection is responded to locally	FALSE
missed_bytes	Indicates number of bytes missed in content gaps, representative of packet loss	0
protocol	Transport layer protocol of connection	udp
duration	How long connection lasted.	0.00228
conn_state	State of the connection	SF
dest_ip	IP address of packet receiver	143.88.5.1
orig_pkts	Number of packets originator sent	2
community_id	Connection's 4-tuple of endpoint addresses/ports	1:Z2qpnUv+rxq4N1rn7Go962U/gi8=
resp_ip_bytes	Number of IP level bytes responder sent	186
dest_port	Incoming port number	53
orig_bytes	Number of payload bytes originator sent. For TCP might be inaccurate	130
local_orig	Records if the connection is originated locally	FALSE
datetime	Datetime of first packet	2022-02-10T03:58:29.979Z
history	Records the state history of connections as a string of letters	Dd
resp_bytes	Number of payload bytes responder sent	130
uid	Unique identifier of connection	CwO2bA321vyBxBjtxb
src_port	Outgoing port number	36073
ts	Time of first packet	1644465509.979958
src_ip	IP address of packet sender	143.88.5.12
mitre_attack_tactics	The MITRE ATT&CK Tactic the row is labeled with	Reconnaissance

Table 4.1: Columns in the dataset with explanations and example data.

Feature selection

The relevance of the available features was assessed by [2] by examining the information gain of the different features. *Information gain* is the difference between a class's entropy

Tactic	Count	Explanation
none	9281599	Benign connections
Reconnaissance	9278722	Gathering information for future operations
Discovery	2086	Figuring out the environment
Credential access	31	Stealing of account names and passwords
Privilege escalation	13	Gaining higher-level permissions
Exfiltration	7	Stealing of data
Lateral movement	4	Moving through the environment
Resource development	3	Establishing resources to support operations
Defense evasion	1	Avoidance of detection
Initial access	1	Initial access to the environment
Persistence	1	Maintain foothold in the environment
Execution	0	Execution of malicious code
Collection	0	Gathering data of interest
Command and Control	0	Communication with compromised systems to control them
Impact	0	Manipulation, interruption, or destruction of systems and data

Table 4.2: Counts and explanations of the tactic labels in the dataset

and the entropy of the class and a selected feature split, with entropy measuring the extent of randomness in the dataset. It is a measure of how useful the feature is in classification. The information gain was calculated the following way for each feature:

$$Gain(A) = Info(D) - Info_A(D)$$

where

$$Info(D) = -\sum_{i=1}^{M} p_i log_2(pi)$$
$$Info_A(D) = \sum_{j=1}^{V} \frac{|D_j|}{|D|} \times Info(D_j)$$

where

- $Info(D)$ is the average amount of information needed to identify the class of a tuple in D,
- $Info_A(D)$ is the expected amount of information needed to classify a tuple from D based on partitioning from A,

- p_i is the nonzero probability that an arbitrary tuple belongs to a class, and
- $\frac{|D_j|}{|D|}$ is the weight of the partition

For the purposes of this work, the six features with the largest information gain were selected, those being *history*, *protocol*, *service*, *orig_bytes*, *dest_ip*, and *orig_pkts*.

Preprocessing

Nominal-valued features are those that contain non-numeric data. These were labeled with integers using scikit-learn. That is, each unique value is given its own label. The number of labels for each of the nominal-valued features is presented in Table 4.3

Feature	Number of labels
service	17
protocol	3
history	241
mitre_attack_tactics	11

Table 4.3: Number of labels of nominal-valued features.

orig_bytes and *orig_pkts* are*continuous-valued* features, meaning that their values represent a numerical value, instead of a category. These features were binned into bins according to value ranges, which are presented in Tables 4.4 and 4.5. The *orig_pkts* feature is divided into bins reflecting different lengths of sessions, and for the (*orig_bytes*) the value ranges reflect the various types of network packets:

- Null values and zero values have their own bins.
- The 1-78 byte range includes normal TCP SYN- and ACK-packets, and normal ICMP-packets.
- The 79-156 byte range includes mainly DNS-packets and some small HTTP-packets.
- Packets larger than 156 bytes are in their own bin.

Bin value range	Number of rows in bin
Null values	776843
0 bytes	9636623
1-78 bytes	4917733
79-156 bytes	6479132
More than 156 bytes	841516

Table 4.4: orig_bytes bins

Bin value range	Number of rows in bin
0 packets	1161805
1 packet	681537
2 packets	10989337
3 packets	2325507
4 packets	7056123
More than 4 packets	437538

Table 4.5: orig_pkts bins

IP-addresses were binned in a special way, to reflect the significance of different classes of IP-address spaces. These are presented in Table 4.6. The address ranges are presented in the way they are present in the dataset - only those IPv6 first group values are listed in the table that appear in the dataset.

IP class	Address range	Number of rows in bin
IPv4 Class A	First octet value 0–126	10120285
IPv4 Class B	First octet value 128–191	12243860
IPv4 Class C	First octet value 192–223	108345
IPv4 Class D	First octet value 224–239	15075
IPv4 Class E	First octet value 240–254	2956
IPv6 Local link	First group value fe80	9131
IPv6 Global internet	First group value 2001 and ff02	152195

Table 4.6: IP-address bins.

Training

A Naïve-Bayes classifier and a Random forest classifier were trained using scikit-learn. The dataset was split into a training-set and a validation-set using the *train_test_split* function belonging to the library, with a randomly-selected 33 percent of the samples being designated as the validation set. Random state of 125 was used.

The classifiers were evaluated by the metrics described in Section 4.3, using the *classification_report* function belonging to the library.

4.3 Model comparison and metrics

There are several different metrics for evaluating classification methods - several different ones are usually used together [13]. In a *multi-class* classification, such as in this experiment, there are multiple different classes a sample can belong to, and to which it can be classified. Table 4.7 shows the *confusion matrix* for a three-class classification problem.

		ACTUAL		
		A	B	C
PREDICTION	A	TP_A	E_{BA}	E_{CA}
	B	E_{AB}	TP_B	E_{CB}
	C	E_{AC}	E_{BC}	TP_C

Table 4.7: Confusion matrix for a classification problem with three classes

The metrics are calculated using values derived from this matrix:

- TP_A denotes the number of *true positives* (TP) - the samples that were classified correctly as class A.
- E_{BA} and E_{CA} denote the number of samples that were incorrectly classified as belonging to class A, even though they belong to classes B and C respectively. The sum of these is the number of *false positives* for class A:
  $FP_A = E_{BA} + E_{CA}$
- E_{AB} and E_{AC} denote the number of samples that were incorrectly classified as belonging to the other classes, even though they belong to class A. The sum of these

is the number of *false negatives* for class A:
$FN_A = E_{AB} + E_{AC}$

Using these values we can calculate the following metrics:

- *Accuracy* is the ratio of correctly classified samples to total samples

$$Acc = \frac{TP_A + TP_B + TP_C}{Total} \tag{4.1}$$

- *Precision* is the ratio of true positives to the sum of true and false positives of a class

$$Precision_A = \frac{TP_A}{FP_A + FN_A} \tag{4.2}$$

- *Recall* is the ratio of true positives to all samples classified as belonging to the class (the sum of true positives and false negatives)

$$Recall_A = \frac{TP_A}{TP_A + FN_A} \tag{4.3}$$

- F_1-score is the harmonic average of precision and recall

$$F_{1A} = \frac{2 \times Precision_A \times Recall_A}{Precision_A + Recall_A} \tag{4.4}$$

The selection of suitable metrics depends on the properties of the data [13]. Different metrics are sensitive to data imbalance, when some classes in a dataset outnumber others. For example, accuracy is dependent on the ratio of positive samples to negative, and thus high accuracy for large classes can hide low accuracy for small ones. When it is important to have good performance for all classes, like in our case, one should not solely rely on accuracy, but instead the class-wise precision and F_1-scores should be considered. A good classifier is one that has good performance for all classes.

5 Results

The performance of the trained models are presented in this chapter.

5.1 First approach

In the first training approach the entire dataset was used, as described in Chapter 4. The accuracy of the Naïve Bayes classifier was 0,82. The other metrics are presented in Table 5.1, and the confusion matrix of the results are presented in Table 5.2.

Metric	Discovery	Reconnaissance	None
Precision	0	0,96	0,95
Recall	0,98	0,66	0,97
F_1-score	0	0,78	0,96

Table 5.1: Metrics of Naïve Bayes classifier

		ACTUAL		
		Discovery	Reconnaissance	None
PREDICTION	Discovery	**669**	875131	230
	Reconnaissance	16	**2018480**	85441
	None	1	168075	**2977552**

Table 5.2: Confusion matrix of Naïve Bayes classifier

The accuracy of the Random Forest classifier was 0,99. The other metrics are presented in Table 5.3, and the confusion matrix of the results are presented in Table 5.4.

Metric	Discovery	Reconnaissance	none
Precision	0	1	0,99
Recall	0	0,99	1
F_1-score	0	0,99	0,99

Table 5.3: Metrics of Random Forest classifier

PREDICTION \ ACTUAL	Discovery	Reconnaissance	None
Discovery	**0**	0	0
Reconnaissance	684	**3018517**	11959
None	2	43169	**3051264**

Table 5.4: Confusion matrix of Random Forest classifier

The results were poor, since the models were biased towards the classes with larger sizes. Even though accuracy was high for both classifiers, the F_1-scores and precision for the Discovery-class were zero or close to zero in both cases. The classifiers were unable to accurately predict the Discovery-samples, but they still had good accuracy-metrics since the data is heavily imbalanced. To rectify this deficiency, a second approach was done, where class weights are adjusted.

5.2 Second approach

For the second approach a balanced subset from the dataset was used. The subset had the same number of samples from each class: it included the 2086 samples of the Discovery class, and randomly selected 2086 samples from the Reconnaissance- and none-classes each.

The accuracy of the Naïve Bayes classifier was 0,87. The other metrics are presented in Table 5.5, and the confusion matrix of the results are presented in Table 5.6.

Metric	Discovery	Reconnaissance	None
Precision	0,78	0,91	0,95
Recall	0,97	0,67	0,96
F_1-score	0,87	0,77	0,95

Table 5.5: Metrics of Naïve Bayes classifier trained on balanced classes

The accuracy of the Random Forest classifier was 0,9. The other metrics are presented in Table 5.7, and the confusion matrix of the results are presented in Table 5.8.

		ACTUAL		
		Discovery	Reconnaissance	None
PREDICTION	Discovery	**671**	187	0
	Reconnaissance	18	**461**	25
	None	0	38	**666**

Table 5.6: Confusion matrix of Naïve Bayes classifier trained on balanced classes

Metric	Discovery	Reconnaissance	none
Precision	0,78	0,98	0,98
Recall	1	0,7	0,99
F_1-score	0,88	0,82	0,98

Table 5.7: Metrics of Random Forest classifier trained on balanced classes

		ACTUAL		
		Discovery	Reconnaissance	None
PREDICTION	Discovery	**686**	190	2
	Reconnaissance	2	**481**	6
	None	1	15	**683**

Table 5.8: Confusion matrix of Random Forest classifier trained on balanced classes

6 Discussion

The results of the first approach were not good. The data was heavily imbalanced, even though only the three largest classes were used: the Reconnaissance- and none-classes were both three orders of magnitude larger than the Discovery-class. This caused both classifiers to perform poorly with the Discovery-class. The Random Forest classifier had zero true positives, and therefore zero precision and F_1-score. The Naïve Bayes classifier had 669 true positives for the Discovery-class, but since it had 875361 false positives, the precision and F_1-score are close to zero.

The result can be understood from the way the models work. For Naïve Bayes the relative sizes of the classes in the dataset provide the prior $P(C_k)$ that is then multiplied by the conditional probability of the new event. With imbalanced data the priors for the larger classes become overpowering, so the smaller classes rarely get selected as the result. For the Random Forest something similar happens, since a majority of the decision trees are likely to vote for the larger classes. In the case of the Random Forest, it is possible to tweak the algorithm to penalize for misclassifying the smaller classes, but that was not investigated in this thesis.

The second approach worked significantly better. For the Naïve Bayes classifier the lowest F_1-score was 0,77 and for Random Forest 0,82. They had similar results in many ways: in both classifiers misclassification of Discovery-samples as belonging to the Reconnaissance-class was an order of magnitude larger than any other type of misclassification; whereas misclassifications between the Discovery- and none-classes was rare both ways.

The results of the first approach were expected: the classifiers used in the experimentation were simple, and therefore their performance is limited when data is imbalanced. Overall the best performance was exhibited by the Random Forest classifier trained in the second approach. It was especially good at classifying between benign and intrusion samples, as it had only 16 false positives and 8 false negatives out of 2086 positive and 4172 negative samples for the none-class.

The severe imbalance of the original dataset resembles real-life data. Intrusion attempts are rare, and intrusion tactics other than reconnaissance are even rarer. Therefore, creating balanced labeled datasets that the techniques described in this thesis require, for other tactics besides reconnaissance and discovery, might be challenging.

7 Conclusions

Malicious intrusions into computer systems are a large and growing problem globally, and detecting them in time plays a key part in preventing the damage they cause. Traditional intrusion detection mechanisms are supplemented by new mechanisms based on machine learning methods, which can help in reducing false positives and broadening detection scope.

In this thesis a comparative study of two machine learning classifier models — Naïve Bayes and Random Forest — for intrusion detection was conducted, using the UWF-ZeekData22-dataset for training. Two approaches were used: the first using the entire dataset, and the second on a balanced subset of the data.

The approach using balanced data worked markedly better, which was expected when using simple classifiers. The first approach did not work at all, since the F_1-score for one class was zero. Random forest performed better than Naïve Bayes in the second approach.

Future work

Training more complex models for intrusion detection is an obvious direction of future work. The models implemented in this thesis are quite simple, so better performance could probably be attained by implementing deep learning models.

A notable challenge in developing supervised learning models for intrusion detection is the availability of suitable labeled datasets. Development of ways to collect or create labeled data would be beneficial.

Generative Adversarial Networks is a promising deep learning framework for this type of problem. Once trained, the generator can be used to create fake samples of rare types of intrusion attempts, and the discriminator can be used as a classifier.

This thesis focused on traditional computer networking, but future work would be useful on intrusion detection in other types of networks, such as Internet of Things (IoT), operational technology (OT) and mobile networks. Both the benign network data, and the intrusion attempts are different in these technologies, so the work in this thesis is not directly applicable.

Bibliography

[1] J. P. Anderson. “Computer Security Threat Monitoring and Surveillance”. 1980.

[2] S. Bagui, D. Mink, S. Bagui, T. Ghosh, T. McElroy, E. Paredes, N. Khasnavis, and R. Plenkers. “Detecting Reconnaissance and Discovery Tactics from the MITRE ATT&CK Framework in Zeek Conn Logs Using Spark’s Machine Learning in the Big Data Framework”. In: *Sensors (Basel, Switzerland)* 22.20 (2022), pp. 7999–. ISSN: 1424-8220.

[3] S. S. Bagui, D. Mink, S. C. Bagui, T. Ghosh, R. Plenkers, T. McElroy, S. Dulaney, and S. Shabanali. “Introducing UWF-ZeekData22: A Comprehensive Network Traffic Dataset Based on the MITRE ATT&CK Framework”. In: *Data (Basel)* 8.1 (2023), pp. 18–. ISSN: 2306-5729.

[4] J. O. Berger. *Statistical decision theory and Bayesian analysis*. 2nd ed. Springer series in statistics. New York, NY: Springer, 1985. ISBN: 0-387-96098-8.

[5] W. E. Forum. “The Global Risks Report 2023”. https://www.weforum.org/reports/global-risks-report-2023. [Online; accessed 14-May-2023]. 2023.

[6] T. Hastie. *The elements of statistical learning : data mining, inference, and prediction*. 2nd ed. Springer series in statistics. New York: Springer, 2009. ISBN: 0-387-84857-6.

[7] E. M. Hutchins, M. J. Cloppert, and R. M. Amin. “Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains”. https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/LM-White-Paper-Intel-Driven-Defense.pdf. [Online; accessed 10-April-2023]. 2011.

[8] H. Liu and B. Lang. “Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey”. In: *Applied sciences* 9.20 (2019), pp. 4396–. ISSN: 2076-3417.

[9] M. N. Murty. *Pattern Recognition An Algorithmic Approach*. 1st ed. 2011. Undergraduate Topics in Computer Science. London: Springer London, 2011. ISBN: 0-85729-495-4.

[10] S. J. Russell. *Artificial intelligence : a modern approach*. 3rd ed. Upper Saddle River, N.J. ; Pearson Education, 2010. ISBN: 0-13-207148-7.

[11] "Scikit-learn: Machine Learning in Python". In: *Journal of machine learning research* (2011). ISSN: 1532-4435.

[12] B. E. Stromand, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, and C. B. Thomas. "MITRE ATT&CK: Design and Philosophy". https://attack.mitre.org/docs/ATTACK_Design_and_Philosophy_March_2020.pdf. [Online; accessed 23-March-2023]. 2020.

[13] A. Tharwat. "Classification assessment methods". In: *Applied computing & informatics* 17.1 (2021), pp. 168–192. ISSN: 2210-8327.

Appendix A Kandidaatin tutkielman tiivistelmä

A.1 Johdanto

Tunkeutuminen tarkoittaa pääsyn saamista tietojärjestelmiin tai niiden toiminnan haittaamista ilman lupaa tai muuta perustetta [8]. Ne aiheuttavat suura kustannuksia yhteiskunnalle, ja yksityisten tietojen vuotaminen aiheuttaa kärsimystä yksilöille. Tunkeutumisen havaitsemisjärjestelmä on ohjelmisto jonka tarkoituksena on suojella tietojärjestelmää havaitsemalla tunkeutumisyritykset. Varhaiset tunkeutumisen havaitsemisjärjestelmät luotiin tietokonevirusten havaitsemiseen, joten ne eivät ole tehokkaita nykyaikaisten, kohdistettujen hyökkäysten torjumisessa [7]. Nykyaikaisten hyökkäysten mallintamiseen on luotu erilaisia viitekehyksiä, joista tässä työssä käytetään MITRE Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK) -kehystä. ATT&CK-viitekehyksessä tunkeutumisen elinkaaren eri osia mallinnetaan taktiikoina, tekniikoina ja alitekniikoina [12]. Taktiikat kuvaavat erilaisia lyhyen aikavälin päämääriä, joita yritetään saavuttaa osana tunkeutumista; ja tekniikat kuvaavat erilaisia tapoja näiden päämäärien saavuttamiseen. Alitekniikat ovat hienojakoisiempia tekniikoiden alatyyppien kuvauksia.

A.2 Tunkeutumisen havaitsemisjärjestelmät

Tunkeutumisyritysten havaitsemiseksi havaitsemisjärjestelmät tarkkailevat erilaisia dataobjekteja [8], ja järjestelmiä voidaan luokitella niiden käyttämien datalähteiden perusteella: tietoverkkotunkeutumisen havaitsemisjärjestelmät analysoivat verkkoliikennettä ja laitetunkeutumisen havaitsemisjärjestelmät analysoivat käyttöjärjestelmien ja ohjelmistojen lokitiedostoja. Perinteisesti tunkeutumisyrityksiä on pyritty tunnistamaan joko erilliseen tietokantaan talletettuihin tuntomerkkeihin perustuen, tai etsimällä datasta poikkeamia normaalitilasta. Kumpaankin tapaan liittyy merkittäviä puutteita: tuntomerkkeihin perustuva tunnistaminen vaatii laajamittaisen tietokannan ylläpitoa, ja sen avulla ei pystytä tunnistamaan uudentyyppisiä tunkeutumisyrityksiä ollenkaan; poikkeamien tunnistaminen puolestaan perustuu luotettavaan normaalitilan määrittelyyn, joka ei useimmiten ole helppoa, ja se tuottaa paljon vääriä hälytyksiä. Koneoppimismallit ovat lupaava teknologia näiden puutteiden lievittämiseen.

A.3 Koneoppimismetodit

Koneoppimismalleja voidaan kategorisoida kahdella eri tavalla: ne jakaantuvat ohjattuja tai ohjaamattomia malleihin; ja ne jakaantuvat perinteisiin koneoppimisalgoritmeihin, ja syväoppimismallihin. [8]. Perinteiset koneoppimismallit ovat olleet tutkimuksen kohteena jo pitkään - ne ovat syväoppimismalleja yksinkertaisempia, eikä niiden suorituskyky ole usein yhtä hyvä. Syväoppimismallit perustuvat neuroverkkoihin, mikä tekee niistä monimutkaisia, raskaita luoda ja ajaa, sekä vaikeampia ymmärtää. Toisaalta niiden suorituskyky on merkittävästi parempi kuin perinteisillä malleilla. Ohjatussa oppimisessa opetusaineisto koostuu syötteistä ja tuloksista, jotka syötteistä tulisi seurata [8]. Tavoitteena on, että luokittelija oppii päättelemään miten tulos seuraa syötteestä, ja sitä voidaan sen jälkeen käyttää uuden aineiston luokitteluun. Tässä työssä käytetään kahta perinteistä ohjatun oppimisen luokittelijaa: Naiivi Bayes- ja satunnaismetsä-malleja.

Naiivi Bayes -luokittelija perustuu Bayesin teoreemaan, joka on tapa estimoida todennäköisyyksiä perustuen aiemmin tunnettuun ennakkotietoon ja havaintoaineistoon [9]. Luokittelijassa lasketaan todennäköisyydet jokaiselle luokalle käyttäen opetusaineistoa havaintoaineistona ja luokkaa ennakkotietona. Luokittelussa luokka, jonka todennäköisyys on annetulla näytteellä suurin, valitaan näytteen luokaksi.

Satunnaismetsäluokittelija perustuu toiseen luokittelijamalliin, päätöspuuhun. Päätöspuun opetuksessa opetusaineisto jaetaan toistuvasti osajoukkoihin jonkin jakotavan perusteella. Jakaminen tehdään puun lehdessä jonkin opetusaineiston komponentin perusteella, ja se johtaa alilehtiin, joissa jakaminen tehdään jonkin toisen komponentin perusteella. Tätä prosessia jatketaan kunnes lehdessä on enää ainoastaan yhtä luokkaa edustavia näytteitä, tai kunnes jakaminen ei enää paranna luokittelua. Satunnaismetsäluokittelijassa opetetaan suuri määrä päätöspuita opetusaineiston eri osilla, ja luokka valitaan sen perusteella, mitä enemmistö päätöspuista antaa näytteen luokaksi.

A.4 Tutkimusmetodit ja tulokset

Käytimme opetusaineistona *UWF-ZeekData22*-tietoaineistoa, johon on kerätty verkkoliikennettä tietoturvakoulutusympäristöstä [3]. Aineistossa on erilaisia tunkeutumisyrityksiä, jotka on luokiteltu MITRE ATT&CK-viitekehyksen mukaan, sekä tavallista verkkoliikennettä. Aineistosta valittiin aiemman tutkimuksen pohjalta ne kuusi komponenttia, joilla on korkein informaatiohyöty, ja ne kolme viitekehyksen luokkaa joilla oli suurin

määrä näytteitä: Discovery, Reconnaissance sekä none, joista jälkimmäisin kuvaa tavallista verkkoliikennettä. Diskreeteille komponenteille annettiin kokonaislukutunnukset, ja jatkuva-arvoiset komponentit lokeroitiin niiden ominaisuuksien perusteella. Tulosten vertailuun käytettiin eri luokkien F_1-arvoja.

Ensimmäisessä yrityksessä mallien opetukseen käytettiin koko tietoaineistoa. Tulokset olivat huonoja molempien mallien osalta. Vaikka tulosten ulkoinen tarkkuus oli korkea, se johtui tietoaineiston epätasapainosta: muut viitekehysluokat ovat monta kertaluokkaa suurempia kuin Discovery. Hyvä tulos muiden luokkien kohdalla peitti alleen sen, että Discovery-luokan F_1-arvo oli nolla.

Toisessa yrityksessä tietoaineistosta valittiin tasapainoinen otos, jossa kaikkia viitekehysluokkia edustavia näytteitä oli yhtä monta. Tulokset olivat tässä yrityksessä huomattavasti paremmat: Naiivi Bayes -luokittelijan kohdalla huonoin F_1-arvo oli 0,77 ja satunnaismetsän kohdalla 0,82.

A.5 Johtopäätökset

Ensimmäisen yrityksen tulokset olivat odotettuja — kokeissa käytetyt perinteiset mallit ovat alttiita epätasapainoisen opetusaineiston aiheuttamalle vinoumalle. Tämä on seurausta niiden toimintatavoista, joka ei ota huomioon epätasapainoisen aineiston vaikutusta laskelmissa käytettyihin kertoimiin. Toisen yrityksen tulokset olivat parempia, etenkin satunnaismetsän osalta, jossa etenkin tavallisen verkkoliikenteen ja tunkeutumisyritysten erottelu oli tarkkaa.

Tunkeutumisyritysten havaitseminen on tulevaisuudessa yhä tärkeämpää, kun digitalisaation myötä yhä suurempi osa ihmisten kanssakäymisestä ja taloudesta tapahtuu tietoverkoissa. Koneoppimismallit ovat lupaava teknologia tähän tarkoitukseen. Syväoppimismalleilla olisi todennäköisesti mahdollista saada parempia tuloksia kuin tässä työssä saatiin. Etenkin generatiivinen kilpaileva verkosto olisi mielenkiintoinen lähestymistapa tähän ongelmaan, sillä siihen kuuluva luova neuroverkko voisi luoda keinotekoista dataa tulevaa kehitystyötä varten. Valmiiksi luokilteltu opetusdata on hankalaa ja kallista kerätä, ja sen puute on usein rajoittavana tekijänä käytännön sovellusten luomisessa. Tulevaa tutkimusta voitaisiin tehdä myös muunlaisista ympäristöistä, kuten IoT- ja mobiiliverkoista, kerätyllä datalla.