Intrusion Detection by Machine Learning

Dissertation plan

Author: Csaba Brunner
Supervisor: Andrea Kő, Szabina Fodor
# Table of Contents

1. Introduction....................................................................................................................5
2. Literature Review ..........................................................................................................5
   2.1. What is Intrusion Detection? ..................................................................................6
   2.2. Data Mining ...........................................................................................................8
      2.2.1. Supervised learning ......................................................................................17
      2.2.2. Unsupervised learning ...............................................................................24
      2.2.3. Neural Networks .........................................................................................30
      2.2.4. Data Mining utility techniques ....................................................................36
   2.3. Intrusion detection research ....................................................................................43
3. Research overview .........................................................................................................54
   3.1. Research method ..................................................................................................54
   3.2. Context ................................................................................................................54
   3.3. Research goals .......................................................................................................56
   3.4. Research questions ...............................................................................................57
   3.5. Experiment Design ...............................................................................................60
      3.5.1. The decision tree bagging model .................................................................67
      3.5.2. The stacked neural network model ...............................................................70
      3.5.3. Neural networks on TensorFlow and KERAS .............................................74
4. Results ............................................................................................................................76
   4.1. Decision tree bagging results ...............................................................................77
   4.2. Stacking neural network results ...........................................................................78
   4.3. External comparison ............................................................................................80
5. Conclusion .......................................................................................................................81
6. Schedule ........................................................................................................................82
7. Publications .....................................................................................................................83
8. References .......................................................................................................................84
9. APPENDIX .....................................................................................................................90
Figures

Figure 1: Data mining as a multidisciplinary field. Based on: Sharda, Delen and Turban (2018) .............................................................. 9
Figure 2: A simple Taxonomy of Data. Source: Sharda, Delen and Turban (2018) ...................................................... 9
Figure 3: The CRISP-DM process model. Source: Chapman et al. (2000) ......................................................... 12
Figure 4: The SEMMA process model for data mining. Source: Sharda, Delen and Turban (2018) ............................... 14
Figure 5: Knowledge discovery in databases. Source: Fayyad, Piatetsky-Shapiro and Smyth (1996) ....................... 15
Figure 6: Decision Tree. Based on: Han, Kamber and Pei (2011) ............................................................... 18
Figure 7: Optimal separating hyperplane with maximized margin created by SVM. Based on Cortes and Vapnik (1995) ............... 20
Figure 8: KNN classification with k=1. Source: Navlani (2018) ...................................................... 22
Figure 9: K-means clustering algorithm. Source: Piech (2012) ............................................................... 26
Figure 10: Point types in DBSCAN clustering. Source: Lutins (2017) ............................................................... 28
Figure 11: Simple mathematical model for a neuron. Source: Own edit based on Russel and Norwig (2010).............................. 31
Figure 12: architecture of a multilayer feedforward neural network. Source: own edit. 32
Figure 13: Architecture of an autoencoder network. Source: Own edit ...................................................... 36
Figure 14: Bagging classifier process. Source: Budzik (2019) ............................................................... 37
Figure 15: Boosting classifier process. Source: Budzik (2019) ............................................................... 38
Figure 16: Stacking classifier process with linear results combination. Source: Budzik (2019) ...................................................... 39
Figure 17: ROC curve. Source: scikit-learn developers (2018) ............................................................... 41
Figure 18: Classification of network anomaly detection methods. Source: Bhuyan, Bhattacharyyya and Kalita (2014) ............................................................... 46
Figure 19: Relationship between Detection Methods. Source: Ippoliti (2011) ...................................................... 47
Figure 20: Machine learning approaches in intrusion detection. Coverage of Buczak and Guven (2015) ............................................................... 47
Figure 21: The methodological abstraction levels followed in this dissertation. Source: Own Edit. ............................................................... 54
Figure 22: Types of Hybrid intrusion detection. Source: Dua and Du (2016) ............................................................... 59
Figure 23: KDD Cup 1999 class distribution on a 10% sample. Source: Own edit. ...................................................... 64
Figure 24: Iterations on the studied detection model. Source: Own edit ............................................................... 67
Figure 25: Data preprocessing for the detection model prototype. Source: Own edit. ...................................................... 67
Figure 26: The model creation and prediction process of the prototype intrusion detector. Source: Own edit ............................................................... 69
Figure 27: Data preprocessing for the neural network stacking detector. Source: Own Edit. ............................................................... 71
Figure 28: The model creation and prediction process of the stacked neural network model. Source: Own edit ............................................................... 72
Figure 29: The model creation and prediction process of the stacked neural network model backed by TensorFlow and KERAS. Source: Own edit ............................................................... 75
Tables

Table 1: Data Mining Tasks, Methods and Algorithms. Based on Sharda, Delen and Turban (2018) ................................................................. 11
Table 2: Confusion matrix for classifier performance. Source: Han, Kamber and Pei (2011) .................................................................... 40
Table 3: Statistics of redundant records in the KDD train set. Source: Tavallaee et al. (2009) ........................................................................ 65
Table 4: Statistics of redundant records in the KDD test set. Source: Tavallaee et al. (2009) ........................................................................ 66
Table 5: Sampling setup of the prototype intrusion detector. Source: Brunner (2017) ................................................................. 68
Table 6: Presampling class fractions. Source: Own Edit .............................................................................................................. 72
Table 7: aggregated measurement levels for the two-class classification. Based on Brunner (2017) ........................................................................ 77
Table 8: aggregated measurement levels for the five-class classification. Based on Brunner (2017) ........................................................................ 77
Table 9: Sampling validation results. Source: Own edit .............................................................................................................. 78
Table 10: Hyperparameter settings, Source: Own edit .............................................................................................................. 79
Table 11: Stacking model accuracy with base model accuracies. Source: Own edit .................................................. 79
Table 12: Macro-averaged precision, recall and F1-score of the stacking model. Source: Own edit ........................................................................ 80
Table 13: Comparison with models from literature. Measure used: Accuracy. Source: Own edit ........................................................................ 80
Table 14: Comparison with models from literature. Measure used: Recall. Source: Own edit ........................................................................ 81
1. Introduction

Since the early 2000s an increase in the number of system intrusions were reported. The most common malicious activities were DDoS, causing significant revenue loss by shutting down online services, and Botnet infections, taking computational resources from legitimate clients and using those resources for carrying out attacks, such as DDoS. Most of these activities have well-defined patterns, thus detecting them is simple enough for a specialized system, particularly when such a system is supported by machine learning algorithms. Despite the increasing frequency of misuse in practice, machine learning is still not widespread enough in IT security. Therefore, I have decided to study intrusion detection systems from a data mining perspective. This involved the setup, parameterization, training and testing of several intrusion detection models, based mostly on neural network architectures.

In the following sections of this dissertation plan, I will introduce the concept of intrusion detection and Intrusion Detection Systems (IDS), the research conducted in this area so far, the machine learning models and techniques that are, or can be used for intrusion detection, and potential research methodology to apply in this area. These sections are then followed by the outline of my research. I attempted to use multiple models as my understanding of intrusion detection grew, going from simple classification ensembles to more complex stacking models and hybrid anomaly-signature detection solutions (the latter is still under development at the time of writing the dissertation plan). Each provided increasingly better results measured primarily in accuracy and recall.

2. Literature Review

When reviewing the available literature, my approach was the following: first I started with the introduction of the core concepts of this dissertation: intrusion detection systems. Then I continued by describing data mining, its key concepts and the different machine learning models, both supervised and unsupervised. The reason for this was that data mining gains an increasing usability in detecting intrusion attempts, and because it is one of the focus areas of my research. Further sections provide information on the overall intrusion detection research, identifying the key literature within the area: McHugh (2000), Stolfo et al. (2000), Tavallaee et al. (2009), Tsai et al. (2009), Ippoliti (2011, 2013) and Buczak and Guven (2015) just to name a few examples. This subsection discusses works on intrusion detection
n using the machine learning approaches outlined beforehand.

2.1. What is Intrusion Detection?

According to Bhuyan, Bhattacharyya and Kalita (2014, pp. 303, 305) “Intrusion is a deliberate and unauthorized attempt to access information, manipulate information and render a system unreliable or unusable. Intrusion itself is a set of actions aimed to compromise the security of computer and network components in terms of confidentiality, integrity and availability”. Intrusion detection is a set of actions to detect such events, to raise alerts, and to provide information to prevent said events. With intrusion detection, the key assumption is that attack behaviors are significantly discernable from normal activities.

Scarfone and Mell (2007) distinguished three detection methods:

- **Signature-based detection**: compare signatures against observed events to identify incidents. Most effective at detecting known attacks, not so much for new attacks or variations of known attacks.

- **Anomaly-based detection**: is the act of comparing normal activity to observed events to identify deviations. This usually involves the creation of profiles representing the normal behavior. Effective at identifying unknown threats, however with many vulnerabilities coming from the act of profiling (malicious activities might get involved in the profiles, accuracy is a challenge and high false positive rates).

- **Stateful Protocol analysis**: the process of comparing profiles of generally accepted definitions of benign protocol activity for each protocol state against observed activity to identify deviations.

Ippoliti (2011) noted the key difference between anomaly and signature-based methods: Anomaly detectors detect what their name suggests: anomalies and not intrusions. After all some legitimate uses may cause alerts in an anomaly detection model, while other carefully constructed attacks might remain hidden (their similarity to normal profiles is high).

According to Scarfone and Mell (2007), the following types of IDS technologies exist:
• **Network based**: Monitoring traffic for network devices or segments. Deployed at the boundary, in proximity to a firewall, VPN servers, remote access servers and wireless networks.

• **Wireless**: monitoring wireless network traffic for possible intrusions.

• **Network Behavior Analysis**: monitoring network traffic to identify unusual traffic flows (like DDoS).

• **Host based**: monitoring the characteristics of a single host system for suspicious activity.

Of the IDS technologies mentioned above, network and host-based approaches are the most researched and mature, but wireless and network behavior analysis are both receiving increased attention as well.

Dua and Du (2016, p. 10) defined intrusions and intrusion detection as “any unauthorized attempt to access, manipulate, modify, or destroy information or to use a computer system remotely to spam, hack, or modify other computers. An IDS intelligently monitors activities that occur in a computing resource, e.g., network traffic and computer usage, to analyze the events and generate reactions”. In an environment prone to intrusions it is always assumed that an intrusion will leave a trace that is clearly discernable from the traces of normal activity. According to detection principles, Dua and Du (2016) differentiated the following tasks in intrusion detection:

• **Misuse / Signature detection**: an IDS triggering method that generates alarms when a known cyber misuse occurs. Known attacks can be detected reliably with low false-positive rates, however they cannot detect new attacks.

• **Anomaly detection**: triggers alarms when the detected object behaves significantly differently from the predefined normal patterns. These techniques are designed to detect patterns that deviate from an expected normal model built for the data. Subsequently they can detect previously unknown attacks at a higher false alarm rate.

• **Hybrid detection**: to improve the techniques of IDSs, researchers have proposed hybrid detection techniques to combine anomaly and misuse detection techniques in IDSs.
- **Scan detection and profiling:** scan detection generates alerts when attackers scan services or computer components before launching attacks. Profiling modules group similar network connections and search for dominant behaviors using clustering algorithms.

Of the listed detection principles, signature, anomaly and hybrid detection methods are the most commonly used, scan detection and profiling, though important on their own, are not part of this dissertation.

### 2.2. Data Mining

To better understand the various detection methodologies used in IDSs, one must understand the core concept, purpose and key categories of data mining. The most prominent definitions for data mining were provided by Han, Kamber and Pei (2011, pp. 26–31) and Sharda, Delen and Turban (2018, p. 222): “extraction – or mining – of hidden knowledge from large amounts of data”. It is a key part of the broader process of Knowledge Discovery in Databases. “Technically, data mining is a process using statistical, mathematical and artificial intelligence techniques to extract and identify useful information and subsequent knowledge from large sets of data”. From the perspective of an intrusion detector, the hidden knowledge is the unknown intent at the source of the network traffic, and the data is the entirety of the inbound network traffic, available in log files for analysis. The goal is to differentiate packets sent with malicious intent from the legitimate ones.

Data mining is a blend of multiple disciplines, including management science & information systems, statistics, artificial intelligence, machine learning, pattern recognition, information visualization, database management and data warehousing (Figure 1). The terms data mining and machine learning are often used interchangeably as synonyms; however, this is not entirely correct. The goal of data mining is to find valuable hidden patterns of information in large amounts of data, as Sharda, Delen and Turban (2018) states. Machine learning, on the other hand, is the application of artificial intelligence to provide systems the ability to automatically learn and improve from experience without being explicitly programmed according to the Expert System Team (2017). The focus of machine learning is on the computer program that can access and use data to learn by themselves. Data mining is interested in the same data to find hidden
patterns. Machine learning algorithms can help with finding these patterns, thus they are frequently used in data mining domains.

To understand data mining and machine learning, first the types of data has to be understood. Sharda, Delen and Turban (2018, p. 87) defined the concept of data as the following: “Data refers to a collection of facts usually obtained as the result of experiments, observations, transactions, or experiences... data are often viewed as the lowest level of abstraction from which information and then knowledge are derived”.

Figure 1: Data mining as a multidisciplinary field. Based on: Sharda, Delen and Turban (2018)

Figure 2: A simple Taxonomy of Data. Source: Sharda, Delen and Turban (2018)
Based on Figure 2, data can be divided into structured, unstructured and semi-structured categories. Unstructured data is created for human observers, thus are more challenging for computers to process. Semi-structured data has some level of data semantics, but the type of data is still varied. Structured data is data stored in tabular format in databases and data warehouses and is the primary type of data used by machine learning algorithms. Structured data is further divided into categorical and numeric data. The taxonomy for categorical and numerical data is:

- **Nominal**: simple codes assigned to entities, the only applicable operation to them is equality testing. A good example for nominal data whether someone smokes or does not smoke.
- **Ordinal**: entities can be ordered into a ranking with data entries. Two ordinal values can be compared whether one is higher than the other, but the magnitude of difference cannot be determined. The most common ordinal data is school grading systems (either 1-5 or A-F).
- **Interval**: values are measured on interval scale, which is defined with an artificial zero value, therefore only the difference between interval values are meaningful, their ratio is not. A typical interval scale is temperature measured in Celsius degrees.
- **Ratio**: continuous values with naturally zero location. The full spectrum of operations is applicable to ratio data. A good ratio data is temperature measured in Kelvin degrees.

The most common types of data mining according to Sharda, Delen and Turban (2018) can be found on Table 1. **Prediction** is commonly referred to as the act of telling about the future. It is not just simple random guessing, as it takes past experiences, opinion and data into account. The two most common methods in prediction are classification and regression. The difference between classification and regression is that classification predicts categorical, while regression predicts numerical data. **Association** discovers interesting relationships between entities in large databases. These entities could be two products that are frequently purchased together. The two methods used for relationship detection are link and sequence analysis. Link analysis does not take the order of precedence between entities into account, whereas sequence analysis does. Finally, with **Segmentation** the goal is to partition and group the data stored in structured form based on a similarity metric. This partitioning includes clustering and outlier analysis. The
former creates homogenous groups where members in one group are more similar compared to members of other groups. Outlier analysis tries to find entities that are more dissimilar to others in general. By excluding these dissimilar entities, the fitness of future data mining processes can be improved.

<table>
<thead>
<tr>
<th>Data Mining Tasks &amp; Methods</th>
<th>Data Mining Algorithms</th>
<th>Learning Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Prediction</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Classification</td>
<td>Decision Trees, Neural Networks, Support Vector Machines, KNN, Naïve Bayes, GA</td>
<td>Supervised</td>
</tr>
<tr>
<td>Regression</td>
<td>Linear/Nonlinear Regression, ANN, Regression Trees, SVM, KNN, GA</td>
<td>Supervised</td>
</tr>
<tr>
<td>Time Series</td>
<td>Autoregressive Methods, Averaging Methods, Exponential Smoothing, ARIMA</td>
<td>Supervised</td>
</tr>
<tr>
<td><strong>Association</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Link analysis</td>
<td>Expectation Maximization, Apriori Algorithm, Graph-Based Matching</td>
<td>Unsupervised</td>
</tr>
<tr>
<td>Sequence analysis</td>
<td>Apriori Algorithm, FP-Growth, Graph-Based Matching</td>
<td>Unsupervised</td>
</tr>
<tr>
<td><strong>Segmentation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clustering</td>
<td>k-means, Expectation Maximization</td>
<td>Unsupervised</td>
</tr>
<tr>
<td>Outlier analysis</td>
<td>k-means, Expectation Maximization</td>
<td>Unsupervised</td>
</tr>
</tbody>
</table>

Table 1: Data Mining Tasks, Methods and Algorithms. Based on Sharda, Delen and Turban (2018)

A different classification of machine learning algorithms is based on the learning process they use, according to Russel and Norwig (2010):

- **Supervised learning** the algorithm observes some input-output pairs and learns a function that maps from input to output. In this setup, the inputs are called independent variables and the output as target variable.

- **Unsupervised learning**, the algorithm learns patterns in the input, even though no explicit feedback is supplied. These algorithms often perform self-organization as part of the learning process.

- **Semi-supervised learning**, the algorithm receives only a few labeled examples, and the agent must make decisions with large amounts of data missing those
labels. The complexity in this case is further increased by systematic noise in the data, as the labels provided might be incorrect themselves.

- **Reinforcement learning**, the algorithm learns from a series of reinforcements: rewards or punishments. Reinforcement learning approaches are uncommon in the data mining literature, thus are not covered by this dissertation.

Out of the data mining methods listed in Table 1, classification, clustering and outlier analysis are the most common in Intrusion Detection research. These methods can be organized into supervised and unsupervised types, the former representing techniques used for signature detection, the latter for anomaly detection, though some overlap between the two exists, for example, support vector machines, traditionally trained in a supervised fashion, can be altered for anomaly detection as a semi-supervised algorithm.

To systematically carry out data mining projects, a general process flow is required. Some of the most popular data mining process models are the Cross Industry Standard Process for Data Mining (CRISP-DM) designed by Chapman et al. (2000), the SEMMA model by Sharda, Delen and Turban (2018) and the Knowledge Discovery in Databases (KDD) of Fayyad, Piatetsky-Shapiro and Smyth (1996).

![The CRISP-DM process model. Source: Chapman et al. (2000)](image)

The CRISP-DM process shown on Figure 3 starts with a good understanding of the business and the associated need for the data mining project and ends with the deployment.
of a solution that satisfies the specified business need. The process itself is highly iterative, and consists of the following steps:

1. **Business understanding**: a key element of any data mining project is finding out what the project is meant to achieve. In this stage business questions are formulated, and a project plan is developed together with assigning the resources and budget needed.

2. **Data understanding**: the next step in the process is to find and understand the relevant data that may come from many data sources. To acquire this understanding, many simple statistical and graphical techniques are used.

3. **Data preparation**: the purpose of data preparation (or data preprocessing) is to take data identified in the previous step and prepare it for the data mining algorithms, for example, by normalizing numerical data.

4. **Model building**: in this step modeling techniques are selected and applied to a cleaned data set to address the needs and answer the questions specified in the business understanding step.

5. **Testing and evaluation**: The developed models are assessed and evaluated for their generalization capability.

6. **Deployment**: model development and assessment are not the end of the data mining project. The knowledge gained must be organized and presented in a way that the end user will be able to benefit from. The models are not trained for the sake of creation, but they are used to increase productivity in some way, therefore these models must be deployed to benefit from insights and lessons learned.

In addition to CRISP-DM there are two additional process methodologies for data mining. The first one is SEMMA, visible on Figure 4. It is an acronym standing for Sample, Explore, Modify, Model and Assess. It begins with a representative sample of the data, makes it easy to apply exploratory statistical and visualization techniques, select and transform the most important predictive variables, model them to predict outcomes and confirm a model’s accuracy. The main difference between CRISP-DM and SEMMA is that CRISP-DM takes a more comprehensive approach to the data mining process, including both business and data understanding in the process. SEMMA implicitly assumes that the same business understanding has been achieved before the data mining process.
The third process model, KDD (Figure 5), uses data mining methods to find useful information in data. KDD is a more comprehensive process, where data mining is an important step, rather than the focus of the project. The activities of the KDD process:

1. **Data selection**: selection and query of the data for analysis. Also involves data integration, when data from multiple sources are joined together.

2. **Data cleaning and preprocessing**: remove noise and inconsistencies in the data.

3. **Data transformation**: prepare the data for analysis and data mining by performing aggregations and operations on feature columns.

4. **Data mining**: train data mining models to detect hidden patterns in the data.

5. **Interpretation and evaluation**: evaluate the detected patterns to see whether they are interesting for a business perspective, based on a defined metric.
Data mining has become a popular tool in addressing many complex business questions and opportunities. Sharda, Delen and Turban (2018) proved that it can be useful in many areas including:

- **Customer relationship management**: detect the most likely responders and/or buyers of a product, understand the root causes of customer attrition and identify the most important customers.

- **Banking**: automate the loan application process and detect fraudulent credit card and online banking transactions.

- **Retail and logistics**: predict accurate sales volume levels to determine correct inventory levels, identify sales relationship between products to improve store layout and optimize sales promotions and forecast consumption levels of products to optimize logistics.

- **Manufacturing and production**: predict machinery failures before they occur through sensor data and identify anomalies and commonalities in production to optimize capacity.

- **Brokerage and securities trading**: predict when and by how much will bond prices change, forecast range and direction of stock fluctuations and identify and prevent fraudulent activities on the stock market.

- **Insurance**: forecast claim amounts for property and medical coverage costs for better business planning, determine optimal rate plans based on the analysis of
claims and customer data and predict which customers are more likely to buy new policies with special features.

- **Computer hardware and software**: predict disk drive failures well before they occur, identify and filter unwanted web content and e-mail messages and **detect and prevent computer network security breaches**.

- **Government and defense**: forecast the cost of moving military personnel and equipment, predict adversary’s moves and, hence, develop more successful strategies for military engagements and identify classes of unique experiences, strategies, and lessons learned from military operations for better knowledge sharing.

- **Travel industry**: predict sales of different services to optimally price services to maximize revenues, forecast demand at different locations to better allocate limited organizational resources and identify the most important customers and provide them with personalized services to maintain their repeat business.

- **Healthcare and medicine**: identify novel cost-benefit relationships between different treatments to develop more effective strategies, forecast the level and time of demand at different service locations to optimally allocate organizational resources and identify novel patterns to improve survivability of patients with cancer.

- **Entertainment industry**: analyze viewer data to decide what programs to show during prime time and how to maximize returns by knowing where to insert advertisements. Predict the financial success of movies before they are produced to make investment decisions and to optimize the returns and develop optimal pricing policies to maximize revenues.

- **Homeland security and law enforcement**: identify patterns of terrorist behavior, discover crime patterns to help solve criminal cases in a timely manner and **detect and stop malicious attacks on critical information infrastructures**.

- **Sports**: most, if not all, professional sports employ data crunchers and use data mining to increase their chances of victory.
With data mining, organizational data and information became the primary source of competition on a global scale according to Nemati and Barko (2001). Organizations that successfully leverage the decision-enhancing environment realized by data mining can both obtain and maintain a lasting competitive advantage. This is the main strategic benefit of data mining.

This section discusses data mining techniques sorted by type: supervised learning first, followed by unsupervised learning. The only exceptions are neural network algorithms, playing a key role in this dissertation, will be introduced in greater detail in their own subsection, particularly back propagation and autoencoder networks. In the last remaining paragraphs, techniques used to assist the process of model training and evaluation are introduced. These include:

- Ensemble methods combining results from multiple models
- Metrics used to evaluate classification performance,
- Synthetic sampling, SMOTE and
- Hyperparameter optimization.

### 2.2.1. Supervised learning

We call a learning process supervised when the learning algorithm possesses available reference information (classes – attack or normal network traffic for an IDS) available to compare learned patterns with. Based on the learned context, new observations can be predicted belonging to a class with higher probability than just by random guessing. The two types of supervised learning are classification (on categorical data) and regression (on numerical data). Intrusion detection is more focused on predicting the outcome of events, therefore classification is a better fit for this problem statement. Training a classifier model can be time consuming, therefore it is often performed off-line, while application is strictly on-line. However, the greatest challenge with classification lies in the fact that the appropriate class labels must be acquired a priori, which is often a lengthy and tedious task. Typical classification algorithms used for intrusion detection, according to Han, Kamber and Pei (2011), Bodon and Buza (2014) and Dua and Du (2016) are:

**Decision Trees**

Decision trees are sets of hierarchical if-then decisions generated by recursive partitioning algorithms according to a set purity measure. An example decision tree for a hypothetical
credit scoring application can be seen on Figure 6. Represented in tree-like structures, an object can be classified by starting from the root node moving along the edges (~rules) towards the leaves. The final class of the object is given by the label of the leaf node.

Decision trees are constructed by recursive partitioning. The process according to Han, Kamber and Pei (2011) involve the following steps:

- Initially, the tree consists of a single node, the root node.
- If all observations within a node belong to a single class, then this node will become a leaf with the class value as a label.
- Otherwise, an attribute is selected according to an impurity measure. This impurity measure can be the Information Gain ratio based on the Shannon-entropy, or the Gini index. This measure determines which attribute and value is selected for partitioning.
- The sample is then partitioned into subsamples.
- The above steps repeat recursively for each subsample until one of the following stopping criteria is met:
  - All samples on a node belong to a single class. The class will be the label for the leaf node.
If a single attribute can only be selected once and the pool of available attributes for partitioning is empty. In this and the following cases, the label is determined by simple majority vote.

- The number of observations is less than a predefined threshold (priori minimum number of observations rule).
- The number of observations in either node after a split would be smaller than a predefined threshold (posteriori minimum number of observations rule).

The most common algorithms for creating decision trees are ID3 (Interactive Dichotomizer 3) by Quinlan (1986) and CART (Classification and Regression Trees) from Breiman et al. (1984). The main difference between the two is the measure for finding a critical attribute value for partitioning the tree. ID3-based algorithms use Gain ratio and the Shannon-entropy, CARTs prefer the Gini index. Let us take a Y probabilistic variable that takes k possible values with $p_i (i = 1, \ldots, k)$ probability, then the Shannon-entropy of Y will be calculated as

$$H(Y) = H(p_1, ..., p_k) = -\sum_{i=1}^{k} p_i \log_2 p_i$$

Entropy is a core information theoretic concept, it refers to the uncertainty about the value of variable Y. If we observe X probabilistic variable, then the uncertainty of Y will be

$$H(Y|X) = \sum_{\forall x \in X} P(X = x)H(Y|X = x)$$

Therefore, if we observe the values of X, the uncertainty will decrease by

$$I(Y, X) = H(Y) - H(Y|X)$$

Therefore, X provided this much information about the value of Y. This measure is called Information gain. The entropy $H(Y|X)$ has a bias towards attributes with a large number of values (Quinlan (1986)). Information Gain Ratio eliminates this bias by normalizing Information gain with the Entropy of variable X:

$$gain\_ratio(X) = \frac{I(Y, X)}{H(X)}$$
The CART algorithm uses the Gini index, which is formulated as

\[ Gini(P) = 1 - \sum_{i=1}^{k} p_i^2 \]

A key advantage of decision trees is their simplicity for a human observer. A disadvantage is their tendency to overfit; they learn the specific details of the training data they are provided with and generalize poorly on test data. This overfitting is mitigated by pruning. Pruning is the replacement of sub trees in a decision tree to improve predictions on the test set. The two most common methods for pruning are subtree replacement and subtree raising.

**Support Vector Machines**

Support Vector Machines (SVMs) are algorithms used for regression, classification and anomaly detection, designed by Cortes and Vapnik (1995). It constructs an n-dimensional separating hyperplane on n-dimensional data. For an SVM, a separation is considered good, when it has the highest distance (or margin) to the nearest data points, as the higher the margin, the lower the generalization error will be. A 2-dimensional example with optimal margin for SVM can be seen on Figure 7.

![Figure 7: Optimal separating hyperplane with maximized margin created by SVM. Based on Cortes and Vapnik (1995)](image)

Support vector classifiers take training vectors of \( x_i \in \mathbb{R}^p \), \( i = 1, \ldots, n \) in two classes and a vector \( y \in \{1, -1\}^n \). They solve the following problem:
$$\min_{w,b,\xi} \frac{||w||^2}{2} + C \sum_{i=1}^{n} \xi_i$$

Subject to \(y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i\),
\[\xi_i \geq 0, \forall i = 1, \ldots, n\]

Its dual is

$$\min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha$$

Subject to \(y^T \alpha = 0\)
\[0 \leq \alpha_i \leq C, \forall i = 1, \ldots, n\]

Where \(e\) is a vector of ones, \(C > 0\) is the upper bound, \(Q\) is an \(n \times n\) positive semidefinite matrix, \(Q_{ij} = y_i y_j K(x_i, x_j)\), where \(K(x_i, x_j) = \Phi(x_i)^T \Phi(x_j)\) is the kernel function, most commonly linear, though other, more sophisticated kernel functions exist, such as gaussian, radial basis function and sigmoid. \(\Phi\) denotes a function that transforms \(x_i\) vectors into a higher dimensional space. This is often referred to as the kernel trick and is used for training on non-linear data. \(w\) is a weight matrix to define the coefficients of the separating hyperplane, and finally, \(C\) and \(\xi_i\) \((i = 1, \ldots, n)\) are parameters that allow the model to be trained on linearly inseparable data, where \(\xi_i\) defines the distance of \(x_i\) from their respective kernel. With the dual solved, the decision function will be the following:

\[f(x) = \text{sgn}(\sum_{i=1}^{n} y_i \alpha_i K(x_i, x) + \rho)\]

An advantage of SVMs is their simplicity, as they find an optimal separating hyperplane. This hyperplane has been proven to have the highest margin, therefore SVM models tend to generalize well. They are, however less applicable to non-linear or linearly inseparable patterns in the data, though both can be adjusted by using the kernel trick and the \(C\) parameter shown above. The only difficulty is finding the correct value for \(C\). If too large, the model will generalize poorly, if too small, it will have a large error rate. The best strategy for finding \(C\) is to experiment, for example, with hyperparameter optimization and cross validation. A third problem with SVM is that it implies a binary classification problem. This can be mitigated by using either 1 vs 1 or 1 vs rest classification strategies,
meaning k SVM models are trained each comparing two classes, or a selected class and all the remaining data.

**K-Nearest Neighbor**

The k-nearest neighbor algorithm searches the variable space around x’ selected observation and selects the k nearest around it based on Han, Kamber and Pei (2011), Bodon and Buza (2014) and Dua and Du (2016). These will become the k nearest neighbors of x’. Then the target value of x’ (denoted as y’) will be determined as the arithmetic (weighted or non-weighted) mean of the neighboring y values (regression) or by the frequency of y values among the neighborhood (classification). An example of KNN with k = 1 is shown on Figure 8.

![KNN classification with k=1](image)

*Figure 8: KNN classification with k=1. Source: Navlani (2018)*

Some of the key challenges with k-nearest neighbor algorithm is finding the appropriate distance measure and a good k value for separation. The most common answer for the former is the Euclidean distance (provided that both x₁ and x₂ are numerical vectors with length n):

$$\text{Dist}(x_1, x_2) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2}$$

Finding the right value for k is more challenging: a small k provides a good distinction between classes or a more accurate regression, but it is more susceptible for noise in the
data. The best approach for finding \( k \) is trying out multiple settings and choosing the model with the best overall results (optimize the value of \( k \)).

An advantage of KNN is that it is a lazy classifier, therefore model training is fast. Testing and predictions are slower and require more memory, as the complete training data is used by the algorithm. Thus, KNN is less applicable on data that either has too many observations or too many features.

**Bayesian Networks**

Bayesian networks use factored joint probability distributions in a graphical model for decisions about uncertain variables (Han, Kamber and Pei (2011), Bodon and Buza (2014) and Dua and Du (2016)). Bayesian networks rely on the Bayes theorem for classification. Let \( X \) be an observation, \( X_1 \ldots X_n \) be its attributes and \( x_1 \ldots x_n \) its values for each attribute. Let \( Y_j \) mark the probabilistic event that \( X \) belongs to class \( j \), where \( j = 1 \ldots k \). An unknown \( X \) should be assigned to class \( j \) for which \( P(Y_j | X = x) \) is maximized.

According to the Bayes rule:

\[
P(Y_j | X = x) = \frac{P(X = x, Y_j)}{P(X = x)} = \frac{P(X = x | Y_j)P(Y_j)}{P(X = x)}
\]

Where \( P(Y_j | X = x) \) is the posterior probability of \( Y_j \) event (how the likelihood of event \( Y_j \) changed with having information about observation \( x \)), \( P(Y_j) \) is the a priori probability of event \( Y_j \) happening (the likelihood of \( Y_j \) without having information about \( x \)). Similarly, \( P(X = x | Y_j) \) denotes the posterior probability of \( x \) with information about \( Y_j \).

As \( P(X = x) \) is constant for every \( j \) class and \( P(Y_j) \) is either provided a priori or can be estimated from the sample using relative frequencies, the algorithm only needs to maximize \( P(X = x | Y_j) \) in order to maximize \( P(Y_j | X = x) \).

The data needed to calculate every possible \( P(X = x | Y_j) \) posterior probability is often not available in large enough quantity in practice, therefore some versions of Bayesian networks make assumptions about the probabilities to simplify the calculations. For example, Naïve Bayes networks assume the conditional independence of \( X_1 \ldots X_n \). In this case, the \( P(X = x | Y_j) \) probability can be expressed as

\[
P(X = x | Y_j) = \prod_{i=1}^{n} P(X_i = x_i | Y_j)
\]
For k classes. The probabilities $P(X_i = x_i | Y_j)$ can be estimated from the data sample. If $X_i$ is categorical then $P(X_i = x_i | Y_j)$ can be estimated with relative frequencies. When $X_i$ is numerical and the distribution of $P(X_i | Y_j)$ is known, then the probability in question can be determined by estimating the parameters of the distribution with statistical methods.

The most important advantages of Naïve Bayes are its robustness (the models remain stable even if the conditional independence assumption is violated) and theoretical importance (the results of many neural network and curve fitting algorithm equals the maximum likelihood hypothesis provided by the Naïve Bayes algorithm). The disadvantages of Naïve Bayes models are their tendency to lose accuracy when their assumptions (conditional independence and the equal importance of every attribute) are violated. However, when the Naïve Bayes algorithm is combined with attribute selection techniques, then its classifications can rival the performance of decision trees and neural networks.

### 2.2.2. Unsupervised learning

According to Russel and Norwig (2010, p. 694) “In unsupervised learning the agent learns patterns in the input even though no explicit feedback is supplied”. Unsupervised learning is more common in anomaly detection, as they provide stable performance compared to signature detection models, are less costly to train and work on previously unknown patterns as well. However, many unsupervised techniques can only handle numerical attributes, and differentiating the attacks from normal activities is a challenge as well. The two most common types of unsupervised learning are clustering and outlier analysis.

Clustering partitions a collection of entities into segments whose members share a similar characteristic, while members between segments are less likely to share that characteristic. Sharda, Delen and Turban (2018). Many clustering algorithms exist, using different heuristics, therefore they may end up with different clusters even on the same data. Therefore, before putting the results of clustering to use it may be necessary for an expert to interpret, and potentially modify the suggested clusters. The most common types of clustering algorithms are Bodon and Buza (2014):

- **Partitioning** methods divide the data into k disjoint groups (or clusters) so that each group contains at least one observation.
• **Hierarchical** methods construct a hierarchical data structure, commonly referred to as dendrograms.

• **Density-based** methods overcome a common limitation of previously created clustering algorithms, their ability to create elliptical clusters only. For a density-based cluster to be valid, at least k number of observations need to be in a predetermined radius from any observation in the cluster. Apart from clustering, density-based methods can be used for outlier analysis, making them particularly useful for intrusion detection.

Han, Kamber and Pei (2011) and Bodon and Buza (2014) refers to outliers as data unusual and distinctively different from a larger set of observations. Often outliers are the results of errors in recording or of some naturally occurring phenomena. The latter is important, as outliers could indicate fraudulent activities in a banking environment or intrusive behavior in a network. The task of outlier analysis is finding k number of outlier values in a dataset with size of n \((k \ll n)\). This can be broken down to two questions: which observations can be determined to be inconsistent with a large enough portion of the data, and how to find an effective method for detecting them. The most common ways of finding outliers are based on modified classification and clustering methods and on the statistical analysis of observations.

The following paragraphs will describe the most common algorithms used in clustering and outlier analysis based on Han, Kamber and Pei (2011), Bodon and Buza (2014) and Dua and Du (2016):

**K-means clustering**

K-means is the oldest and most common algorithm for clustering. It takes \(n\) observations and partitions them into \(k\) disjoint clusters. Observations in the same cluster are more similar than observations in other clusters. This similarity is based on the arithmetic mean of member observations, often represented as the centroid of a cluster. The goal of k-means clustering is to minimize a predetermined criteria function. The steps performed by the algorithm are shown on Figure 9.
1. Start with the set of observations (Figure 9 (a)). Choose k observations at random (Figure 9 (b)), these will be the first centroids.

2. Assign the remaining \( n - k \) observations to the nearest centroid (Figure 9 (c)), based on distance.

3. Calculate the new centroids for each cluster (Figure 9 (d)).

4. Repeat steps 2 and 3 until the criteria function converges (Figure 9 (e)-(f)). This criteria function is the squared error function:

\[
SE = \sum_{i=1}^{k} \sum_{j=1}^{n} ||p_j - \mu_i||^2
\]

Where \( p_j \) is a point representing an observation and \( \mu_i \) is the arithmetic mean for the \( i^{th} \) centroid.

The k-means algorithm works well when the clusters form compact groups. It is a simple and fast algorithm that scales well with larger datasets. It is, however, not guaranteed to find global optima: it converges on a partitioning, even when a cluster setup could exist with lower squared error. Moreover, the algorithm only works with observations defined in a vector space, therefore categorical features must be excluded or converted to numerical before.

K-means algorithm can be adapted for outlier analysis as well demonstrated by Dua and Du (2016). Without explicitly defining k, the clustering hyperspheres are also constrained...
by a threshold $r$. Given data set $X = \{x_1, \ldots x_n\}$ and cluster set $C = \{C_1, \ldots C_k\}$, the distance $\text{dist}(x_i, C_j)$ measures the similarity between observation $i$ and cluster $j$. The difference from standard k-means algorithm comes when minimal distance between cluster $j$ and observation $i$ is greater than threshold $r$. Then a new cluster is initialized with the observation as its initial centroid. The challenge remaining is to determine which clusters can be considered as normal and which ones as anomalous. The algorithm assumes that normal data outnumbers anomalous data, therefore the cluster that constitutes more than an $\alpha$ percentage of the original training data are labelled as normal, the rest as anomaly.

**Variations of k-means, k-medoid clustering**

Over the years several variations of k-means clustering were designed. These were different in their cluster initialization, the calculation of differences or the calculation of cluster centroids.

One of the more common adjustment to k-means clustering is called k-medoid clustering, aiming to address two disadvantages with k-means: k-medoid results are less sensitive to outliers, and the algorithm uses similarity metrics only, therefore observations are no longer required to be representable in vector spaces. In k-medoid, a cluster is represented not by a calculated centroid value, but by an actual observation as a medoid. Moreover, the criteria function is slightly different; instead of an arithmetic mean, a squared average distance from the medoids ($v_i$) is calculated:

$$SE = \sum_{i=1}^{k} \sum_{j=1}^{n} \| p_j - v_i \|^2$$

The only drawback of the k-medoid algorithm, is that it is less applicable to outlier analysis, as it is less sensitive to outlier values.

**DBSCAN**

DBSCAN is a density-based clustering algorithm, using two parameters ($\varepsilon$, a radius-like parameter and a threshold for the number of observations, called minpts) for determining the density of a cluster developed by Ester et al. (1996). The neighborhood of an observation ($N_\varepsilon(p)$) is the number of observations that fall within an $\varepsilon$ radius around
observation p. To further understand DBSCAN algorithm, one must understand the following definitions:

- Point p is directly density-reachable from point q if \( p \in N_\varepsilon(q) \) (p is within the \( \varepsilon \)-neighborhood of q) and \( |N_\varepsilon(q)| \geq \text{minpts} \) (core point condition). Two core points are density-reachable from each other, a border point is directly density-reachable from a core point, but a core pint is not directly density reachable from a border point.

- Point p is directly density-reachable from point q if there is a chain of points \( \{p_1, \ldots, p_n\} \) \( p_1 = p, p_n = q \) and \( p_{i+1} \) is directly density reachable from point \( p_i \).

- A point p is density-connected to a point q if there is a point o such that both p and q are density-reachable from o.

Then, a cluster in DBSCAN can be defined as a set of density-connected point observations. Those observations that were not assigned to any clusters will be considered as noise (or, in the case of outlier analysis, the outliers themselves). Figure 10 demonstrates different types of observations determined by the DBSCAN algorithm.

The DBSCAN algorithm can detect non-elliptical clusters, however, it is highly sensitive to the two input hyperparameters, \( \varepsilon \) and \( \text{minpts} \). To make the model more challenging, finding these optimal parameters may not even be feasible if observation densities within a cluster is not uniform.
One Class SVM

SVM is considered a supervised classification model, however, Schölkopf et al. (2000) proved that it can be modified to perform outlier analysis as well. One class SVM is an algorithm that learns a function with a returned value of +1 in a small region capturing a large portion of data points (called origin) and -1 everywhere else. To separate the data set from the origin, it solves the following quadratic problem:

$$
\min_{w, \xi, \rho} \frac{1}{2}||w||^2 + \frac{1}{\nu n} \sum_{i=1}^{n} \xi_i - \rho
$$

Subject to

$$
(w^T \Phi(x_i)) \geq \rho - \xi_i,
$$

$$
\xi_i \geq 0, \forall i = 1, \ldots, n
$$

Note, that apart from a change to the C parameter from regular SVM to $\frac{1}{\nu n}$, the problem definition remains largely the same. The $\nu \in (0,1)$ parameter is particularly important for the algorithm as it both sets an upper boundary on the fraction of outliers and a lower boundary on the training examples used as support vectors. The decision function can be determined by solving the dual problem:

$$
\min_{\alpha} \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j K(x_i, x_j)
$$

Subject to

$$
0 \leq \alpha_i \leq \frac{1}{\nu n}, \sum_i \alpha_i = 1
$$

Thus, the decision function will take the form of:

$$
f(x) = sgn((w \cdot \Phi(x_i)) - \rho) = sgn(\sum_{i=1}^{n} \alpha K(x, x_i) - \rho)
$$

One class SVM shares most advantages and disadvantages with the original SVM algorithm, the only major difference being that outlier classification is inherently a binary classification problem, therefore one class SVMs do not need to be adjusted for multiclass classification.

Statistical methods for outlier analysis

Statistics-based outlier analysis assumes that the data corresponds to a statistical distribution (for example, normal distribution), and outliers are determined based on how
well they fit the hypothetic distribution using discordance testing. Three information is needed for a good discordance test: the distribution of a feature, the parameters of the distribution, and the expected number of outlier values.

The discordance test states the following null hypothesis:

\[ H_0: o_i \in F, i = 1, \ldots, n \]

With its \( H_0 \), the discordance test measures whether value \( o_i \) is significantly small (or big) compared to a distribution F. Several different statistics can be employed as discordance test, largely depending on the choice of F, as some values could be considered as outlier for one distribution, but not for another. Then for each \( o_i \) value a \( v_i \) statistic and a \( p_i \) value are both calculated based on F. If any of the \( p_i \) values are sufficiently small, then \( o_i \) can be considered discordant while the null hypothesis is rejected.

The greatest disadvantage of statistical outlier analysis, that most tests only evaluate one feature at a time, when for many data mining tasks outliers must be found in a multi-dimensional space. Moreover, some statistical tests require information on the parameters of the data, like its distribution. This can be mitigated by preferring non-parametric tests.

### 2.2.3. Neural Networks

In this section I introduce the most common neural network model, multilayer perceptron, together with feature autoencoder networks, the latter being better suited for outlier detection. This section is based on the work of Rumelhart et al. (1988), Russel and Norwig (2010, pp. 727–737) and Kingma and Ba (2014).

Artificial neural networks are, at their core, mathematical models of the human brain’s activity. They form networks of massively parallel distributed processing units called neurons. The schematic model for one neuron is presented on Figure 11. Each neuron is either connected with input observation values (\( x_i \)) or the outputs of other neurons (\( a^L_{i-1} \), where \( L \) refers to the layer the current neuron is located at). Each connection has a weight \( w^L_{ij} \) associated with it determining the strength of the connection. The first weight refers to the bias value (\( b_j^L \)), with its associated activation equal to one. Activation \( a_j^L \) of neuron \( j \) is calculated by aggregating the products of prior activations and their associated weights (\( z_j^L \)), then transforming the results using an activation function (\( f \)). This activation is propagated towards neurons on layer \( L+1 \).
Mathematically, neurons perform a sum of products between prior activations and their weights, then apply a function on the aggregation:

\[ z_j^l = \sum_{i=0}^{n} a_{l-1}^i w_{lj}^l \]

\[ a_j^l = f(z_j^l) \]

This is called an activation function \( f \) which can take many forms. The most common are sigmoid, tangent hyperbolic, RELU and leaky RELU:

- **sigmoid**: \( f(z) = \frac{1}{1 + e^{-z}} \)
- **tanh**: \( f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \)
- **RELU**: \( f(z) = \max(0, z) \)
- **Leaky RELU**: \( f(z) = \max(\varepsilon z, z); 0 < \varepsilon \ll 1 \)

Each of these activation functions apart from the two RELU functions, map a value from \((-\infty, \infty)\) to \((0, 1)\) and must be differentiable over the same range. Differentiability is an important aspect of activation functions, as it will become clear once back propagation is reached.

In total, two neural network architectures can be developed:

1. **Feed-forward networks**: connections between neurons form a directed acyclic graph. They have no internal state other than their weights.
2. Recurrent networks: feeds output back into its own inputs. Its initial state can depend on prior inputs, making them adept at modeling short-term memory. They are better suited for problems, that possess inherent sequential patterns, for example, text processing and NLP problems rather than intrusion detection (unless the intrusion detector is tasked to evaluate sequences of network packets, which, for some of the common datasets, is not the case). Though interesting, this section will not discuss RNNs any further.

Feed-forward neural networks are structured into layers (Figure 12), collections of neurons taking inputs from neurons in a preceding layer and propagating their output to neurons in the following layer. A layer receiving inputs from the environment is called input layer, a layer propagating its outputs to the environment is called output layer. All the remaining layers between input and output layers are called hidden layers.

Multilayer feed-forward neural networks are surprisingly flexible models. They can support both classification and regression problems. In fact, McCulloch and Pitts (1943) argued that a network constructed by a sufficiently large number of neurons is capable of approximating any desirable function with categorical or numerical output. Because signature detection, is primarily a classification problem, therefore neural networks used for classification will be the primary focus of this section.
One challenge for multilayer feed-forward networks is how to produce more than one output. In this case original expected output vectors are available, marked as $y_i$. To quantify the performance of the network, the activations of the output layer ($a^L_i$) are compared to this expected output. In classification this is performed with the help of two functions, the cross-entropy loss:

$$\text{Loss} = -\sum_{i=0}^{k} y_i \log(a^L_i) + \frac{\alpha}{2} \|W\|^2_2$$

And the SoftMax functions:

$$\text{softmax}(a^L_i) = \frac{e^{a^L_i}}{\sum_{j=0}^{k} e^{a^L_j}}$$

Where $k$ stands for both the number of classes and the number of output neurons. Expected output $y$ is a vector of length $k$, where $y_i = 1$ if and only if the observation represented by $y_i$ belongs to class $i$, otherwise 0. The SoftMax function takes a vector of length $k$ and returns a vector at the same length with class probabilities. The probability located at index $j$ will be the highest, if an observation belongs to class $j$. This way, the loss for one observation can be calculated. To calculate overall loss for multiple (or all) observations, simply average the loss functions for those observations.

One more element in the loss function worth mentioning is $\frac{\alpha}{2} \|W\|^2_2$, commonly referred to as L2 regularization. The purpose of L2 regularization is to penalize weight updates that are too large, thus preventing the neural network model from overfitting the data. This regularization penalty is controlled by parameter $\alpha$.

Learning in a neural network involves finding the minimum of the average loss function. This optimization process involves the iterative incremental adjustment of the weight matrix $W$. This can be achieved by taking the partial derivative of the loss function with regards to the weights, which is simple considering only the output layer, however, the true challenge lies in propagating the loss over to the hidden layers. This challenge has been solved by the introduction of back-propagation.

Back propagation, as the name implies, propagates the loss measured at the output layer towards the input layer. To do this, one must determine the sensitivity of the loss function
to its weights. This is performed iteratively for each weight by applying the chain rule twice:

$$\nabla \text{Loss}_W \leftarrow \frac{\partial \text{Loss}}{\partial w_{j,k}^\ell} = a_{k}^{\ell-1} \cdot f'(z_j^\ell) \cdot \frac{\partial \text{Loss}}{\partial a_j^\ell}$$

Where

$$\frac{\partial \text{Loss}}{\partial a_j^\ell} = \begin{cases} \sum_{j=0}^{n_{\ell+1} - 1} w_{j,k}^{\ell+1} \cdot f'(z_j^{\ell+1}) \cdot \frac{\partial \text{Loss}}{\partial a_j^{\ell+1}}; & \text{if } \ell \text{ is a hidden layer} \\ a_j^\ell - y_j; & \text{if } \ell \text{ is an output layer} \end{cases}$$

With the formula above, the algorithm calculates the gradient vector ($\nabla \text{Loss}_W$), holding information on how much each weight needs to change to minimize the loss function:

$$W_{i+1}^{\ell} = W_i^{\ell} - \eta \nabla \text{Loss}_W^i$$

Where i is the iteration step and $\eta$ is the learning rate. An iteration can be one complete pass over all $(X,y)$ input-output pair. This is computationally expensive, therefore other methods are preferred: minibatch stochastic gradient descent (SGD), where, instead of all the input data, the algorithm uses small sets of the input for weight update. This step is then repeated for all input subsets. A whole pass of the entire input in SGD is referred to as an epoch, which in turn repeats until a set number or convergence is reached. The other improvements to SGD involve Adam (Kingma and Ba (2014)), which introduced adaptive bias-corrected first and second moments to gradient descent for automated weight adjustments. Since then, Adam has become widely applied as a solver for neural network.

Neural networks advantageously operate as universal function approximators. Given enough time and input, they can learn non-linear functions of any complexity.

The disadvantages of neural networks are:

- The algorithm has no guarantees to find global optimum, neural network instances must be trained multiple times with different weight initializations.
- They tend to overfit presented data. This is offset by applying L2 or L1 regularization (or both in an Elastic Net approach).
Neural networks require the tuning of several hyperparameters, such as learning rate, the alpha parameter for L2 regularization, the number of hidden layers and the number of neurons per hidden layer. Hyperparameter search strategies are required to find an optimal value for each.

Neural networks are also sensitive to feature scaling, mitigated by feature normalization.

Neural networks are often too complex for a human observer to understand, they are often referred to as black box systems.

**Autoencoder Networks**

Autoencoder networks, or variational autoencoders are specialized neural network structures that learn the underlying features of a dataset and try to reconstruct the same data as before according to Ranjan *et al.* (2018) and Ranjan (2019). They are particularly useful in finding outlier patterns, therefore they are interesting for anomaly detection as well. The architecture of an autoencoder network is available on Figure 13. The hidden layers are deliberately set up to have a sequentially decreasing number of neurons per layer (encoder architecture) which is then turned around (decoder architecture). This hourglass shape forces the network to compress and reconstruct the information content of the incoming data. Feature space X is simultaneously the input and output for an autoencoder network.
Training the autoencoder is performed using the same backpropagation process as a regular neural network. The differences lie in network architecture and which portions of the data are presented to the autoencoder:

- The data is split to normal and anomalous (~attack) sets.
- Train the autoencoder only using the normal part of the dataset. Thus, the autoencoder learns normal connection features.
- Reconstruction error on new normal connections are expected to be lower, and higher on attack patterns.

Autoencoder networks are neural networks designed to perform anomaly detection. They can operate on their own, or as part of a hybrid anomaly-misuse detection engine.

### 2.2.4. Data Mining utility techniques

Machine learning algorithms form the core set of techniques data scientists use, however, they use other tools to assist them with data processing, model testing and improvement.
For example, model ensembles, synthetic sampling techniques, hyperparameter optimization and model evaluation. This subsection briefly introduces each of these techniques.

**Ensemble methods**

The idea behind ensemble methods is to take multiple machine learning models and combine them to get an overall or aggregate predicting model. This overall predictor performs better than any of the base models. Under ensemble models Budzik (2019) distinguishes the following:

**Bagging:** aims to sample the data set with replacement for an ensemble of models (Figure 14). This sampling process is repeated for each base model, and the final decision is the simple arithmetic mean (or simple majority vote, in case of classification) of base model results. Bagging is most effective with models that have low bias but high variance. One canonical example of bagging classifiers is random forest.

![Bagging classifier process](image)

Figure 14: Bagging classifier process. Source: Budzik (2019)
**Boosting**: performance is improved by concentrating modeling efforts on data that has been predicted incorrectly in previous iterations (Figure 15). A sequence of models is trained where mispredicted observations are weighted heavier than correct ones. Here, results aggregation is calculated as weighted arithmetic mean or weighted voting. Models with low variance and high bias are well suited for boosting, for example, gradient boosting.

![Boosting diagram](image_url)

**Figure 15**: Boosting classifier process. Source: Budzik (2019)

In **Stacking**, base model results are combined using a function, which might be a machine learning model itself (Figure 16). Stacking, compared to boosting and bagging, can reduce model variance and bias at the same time, providing powerful aggregate models for prediction.
Ensemble models can improve results by reducing model variance, bias or both. Therefore, they are useful for creating models with improved classification or regression performance. A drawback of model ensembles is the increased complexity: as multiple models are trained and maintained simultaneously.

**Synthetic sampling methods**

In case of classification, one way to combat imbalanced classes in machine learning is to under-sample the majority class or over-sample the minority class. However, when the level of class imbalance is too great, more sophisticated methods are required. One such method is SMOTE (Synthetic Minority Over-sampling TEstchnique), developed by Chawla *et al.* (2002). SMOTE performs the following activities:

The minority class is over-sampled by taking one minority observation at a time and introducing new synthetic observations along the line segments joining the selected observation and any of the k nearest neighbors from that minority class. More specifically, take a difference between the selected minority observation and one of its nearest neighbors. multiply this difference by a random number between 0 and 1 and add it to the
selected observation. This will result in the creation of a random point along the line of the original observation and its selected nearest neighbor.

An advantage of SMOTE is that it forces the following machine learning model to create larger and less specific decision regions between classes, thus forcing them to generalize better.

Experiments in this dissertation used a modified version of SMOTE, recommended by Nguyen, Cooper and Kamei (2009) called SVMSMOTE. The core purpose is the same, but instead of using the principle of k nearest neighbor it uses the maximum margin classification principle of support vector machines. The benefit compared to SMOTE that it samples only the border regions between majority and minority classes, thus improving model generalization further.

**Evaluation metrics**

It is important to see how different classification models performed, which can be quantified by evaluation metrics. For nearly all the metrics, data is provided by the confusion matrix, available in Han, Kamber and Pei (2011) and also on Table 2. It contains predictions made by the classifier in its rows and the ground truth classes in its columns. The cells mark the true positive, false positive, false negative and true negative predictions.

<table>
<thead>
<tr>
<th>Prediction</th>
<th>Ground Truth</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>+</td>
<td>True Positive (TP)</td>
<td>False Positive (FP)</td>
</tr>
<tr>
<td>-</td>
<td>False Negative (FN)</td>
<td>True Negative (TN)</td>
</tr>
</tbody>
</table>

Table 2: Confusion matrix for classifier performance. Source: Han, Kamber and Pei (2011)

The most common evaluation method based on the confusion matrix is **accuracy**:

\[
\text{Accuracy} = \frac{TP + TN}{(TP + TN + FP + FN)}
\]

This metric is sensitive to high imbalance in target classes, therefore it is not appropriate for evaluating intrusion detection performance, the researcher needs to use other metrics, like **precision**, **recall**, **F₁-score**, receiver operating characteristic (ROC) curve, area under the ROC curve (AUC). Their calculation:
Precision measures the exactness of positive labeling, the coverage of the correct positive labels among all positive-labelled samples. Recall measures the completeness of positive labelling, the fraction of correctly labelled positive samples among all positive samples. The F-measure combines the two metrics into one and assigns a weighted importance to either precision or recall. This importance weight is set to one, meaning precision and recall are equally important.

The efficiency of Intrusion detection problem can be measured by the false positive rate as well, denoting the percentage of observations misclassified as positive over all observations:

\[
\text{False Positive Rate} = \frac{FP}{TN + FP}
\]

The ROC curve visualizes all possible cuts between the positive and negative predictions, based on sensitivity and false positive rate (or 1-specificity) (Figure 17). In a ROC curve a good prediction with a good cut value converges to the top right or the bottom left corner (in the latter case, changing the class labels is a viable option). The diagonal line represents the results of random guessing.

Figure 17: ROC curve. Source: scikit-learn developers (2018)
\[ Sensitivity = \frac{TP}{(TP + FN)} \]
\[ Specificity = \frac{TN}{(TN + TP)} \]

ROC curves are a visualization technique, they are difficult to understand quantitatively. However, a metric can be calculated from ROC curves called AUC which is easy to interpret as a measurement of generalization ability. A 0.5 AUC score denotes random guessing, a value of 1 a perfect score. Anything between can be considered a good performance.

**Hyperparameter optimization**

Many machine learning models require parameters set up prior to training. These parameters are important, as they directly influence the performance achievable by the model. An automated approach for selecting these model parameters are crucial. This process is called hyperparameter optimization and acts as a wrapper method over the regular train-test-evaluate process of machine learning. Claesen and De Moor (2015) identified the following approaches common in hyperparameter optimization:

- **Metaheuristics-based**: uses algorithms that provide satisfactory results in computationally amenable time, compared to exhaustive search strategies. Metaheuristic approaches usually mimic natural phenomena, for example, swarm optimization, ant colony optimization and genetic algorithms.

- **Random search**: randomly searches the hyperparameter space and selects a combination of parameters a set number of times. The final parameter combination is determined by the best model performance. It is faster compared to grid search strategy; however, it does not guarantee an optimal combination of hyperparameters.

- **Grid search**: the hyperparameter space is divided into equally sized segments (~grids) and at each step a parameter combination in the grid is selected and evaluated. Evaluate the complete grid space once. This method guarantees optimal results; however, its time complexity increases exponentially with the addition of new hyperparameters.
• **Bayesian optimization**: a definition for Bayesian optimization was outlined in Snoek, Larochelle and Adams (2012). This approach constructs a probabilistic model for machine learning model performance (often portrayed as a function) and uses this model to make decisions about the next parameter combination to evaluate while also integrating uncertainty. Bayesian optimization essentially uses all information available from previous model training and evaluation steps to choose a new candidate hyperparameter. This requires additional computations, therefore Bayesian optimization is particularly useful for complex machine learning model training, where the extra computations are justified by finding a better parameter combination for the model to train with (for example, with neural networks). Bayesian optimizers calculate expected improvement in the evaluation by balancing out exploration (finding new parameter combinations) and exploitation (searching near good results to find even better ones). One example of Bayesian optimization is gaussian process optimization. Bayesian optimization is faster than grid search strategy and can still find optimal hyperparameter combinations compared to random search.

This concludes the data scientific and machine learning considerations of the dissertation. The following section (2.3) will introduce the pivotal early works of intrusion detection research and reviews the literature conducted on intrusion detection so far.

### 2.3. Intrusion detection research

J. Stolfo *et al.* (2000) discussed the 1998 DARPA dataset for anomaly and misuse detection. They utilized classification, frequent pattern and sequence analysis techniques. Prior to classification, manual and automated feature selections were also performed. By the end of these selection procedures, the connection features were divided into four different categories:

- **Intrinsic features**: features characterizing every network connection.
- **(time-based) traffic features**: the connections that had the same destination host or service as the current connection in the previous 2 seconds.
- **Host/service-based traffic features**: same as above, but instead of a 2 seconds time window, the authors used the previous 100 connections.
- **Content features**: features that describe the content of the connection.
These four feature sets were used in three machine learning models utilizing the RIPPER algorithm for rule construction (RIPPER: a rule induction based “divide and conquer” algorithm). The target variable consisted of 5 classes: DoS (denial of service attacks), R2L (unauthorized access from a remote machine), U2R (unauthorized access to local superuser privileges by a local unprivileged user), PROBE (surveillance and probing), and NORMAL behavior.

The three models created for better predictive performance:

- **The (time-based) traffic model**: containing intrinsic and (time-based) traffic features, best for detecting DoS and PROBING attacks.
- **The host-based traffic model**: containing intrinsic and host-based traffic features, best for detecting slow PROBING attacks.
- **The content model**: containing intrinsic and content features, best for detecting R2L and U2R attacks.

The three models are then combined into a meta-learner, which decides on the best performing models for the given connection. This functionality is very similar to modern ensemble methods.

J. Stolfo *et al.* (2000) also compared the performance of their solution with other participants of the DARPA Intrusion Detection Evaluation Program. Their aggregated model performed better overall, however, all participants, including the authors themselves, struggled to detect R2L attacks, and the author’s model performed poorly with new DoS and R2L attacks on the test set.

As criticism one could say that J. Stolfo *et al.* (2000) did not analyze the distribution of the target variable, and the fact that they only used ROC curves for model performance measurement. Further criticism of the DARPA 1998 line of datasets can be read in McHugh (2000), most of which can be tracked down to the unit of analysis problem: a given attack can be interpreted for a connection package, or for multiple packages over time, as a session. This causes issues with the evaluation method (ROC curves and AUC) used by J. Stolfo *et al.* (2000) and other participants analyzing this dataset during the KDD Cup 1999 competition.

Another criticism affecting the data, is that the underlying taxonomy has been developed from the attacker’s point of view. This provides information for detection algorithms that
may not be available in a real-life scenario. Instead McHugh (2000) proposed a classification based on the protocol layer and the protocols used, or whether a completed protocol handshake is required to carry out the attack.

Attack event distribution was unrealistic as well, which translated over to the KDD Cup 1999 dataset. This has been noted by Tavallaee et al. (2009) as well: some of the key issues with the data involved the creation of artificial data (both normal and attack), unknown number of dropped packets caused by traffic overflow, and vague attack definitions. The largest issue, however was that both the training and test datasets contained many redundant records (78% and 75%, respectively), which caused machine learning algorithms to have biased results. They, instead, proposed a new dataset, the NSL-KDD dataset having better balanced target classes and less observations overall.

The work of J. Stolfo et al. (2000) is pivotal, but not the only one in intrusion detection. The most accepted methods for intrusion detection are artificial neural networks (ANN), support vector machines (SVM) and decision trees (DT). Both ANNs and SVMs perform well as intrusion detectors, however, not without drawbacks: ANNs are time intensive to train and prone to overfitting, and SVMs can only handle binary classification problems.

In their literature review, Tsai et al. (2009) wrote about intrusion detection research between 2000 and 2007. The authors distinguished three approaches:

- **Single**: using only one machine learning algorithm for classification or anomaly detection.
- **Ensemble approaches** using multiple classifier algorithms as weak learners to train a model with better accuracy.
- **Hybrid**: having two functional components, one processing raw data into intermediary results, the other taking these results and determining a final prediction, e.g.: clustering of features prior to classification.

Tsai et al. (2009) reported single classifiers being used the most, however by 2008, hybrid classification techniques started to gain traction. Ensemble models were not analyzed in depth, partly due to a small sample size in the literature sample (only ~11% of the analyzed literature used ensemble models).

Moreover, Tsai et al. (2009) had looked at two additional characteristics of intrusion detection literature: the datasets used and whether feature selection was considered or not. They, as well as Bhuyan, Bhattacharyya and Kalita (2014) determined that most of the
available literature used one of three datasets: the KDD Cup 1999, the DARPA 1998 and the DARPA 1999 datasets, being the few openly available benchmark datasets. As for feature selection, 46.4% of the reviewed literature did, and 53.6% did not attempt to use it. Based on this Tsai et al. (2009, pp. 11998) claimed that “feature selection is not very popular procedure in intrusion detection.“ which in itself is debatable.

Bhuyan, Bhattacharyya and Kalita (2014) identified six different methods used for network anomaly detection: statistical methods, classification, clustering and outlier detection, soft computing, knowledge-based models and combination learners. Out of them, classification, clustering, outlier analysis, soft computing algorithms (specifically neural networks) and combination learners are the most researched areas (Figure 18).

Figure 18: Classification of network anomaly detection methods. Source: Bhuyan, Bhattacharyya and Kalita (2014)

(Figure 19) shows the classification of Ippoliti (2011). Compared to Bhuyan, Bhattacharyya and Kalita (2014), he grouped classification, clustering and outlier analysis under machine learning, distributed elements of soft computing between the remaining four categories and identified knowledge based and combination learners as rule based and hybrid approaches.
Buczak and Guven (2015) discussed the potential of using data mining and machine learning algorithms for Intrusion detection, more particularly for signature detection, anomaly detection and hybrid approaches, the latter two combined into one category due to their low representation in the literature. The covered algorithms can be seen in Figure 20.

The most commonly used metrics for classifier evaluation are detection rate (sometimes referred to as recall or sensitivity), false alarm rate (named false positive rate or 1-specificity in literature), ROC curves & AUC and classifier accuracy. Recall and false positive rate better describe model performance in intrusion detection, as attacks that remain undetected, are just as harmful for an organization, as legitimate connections marked as attacks. The two metrics capture the two classification mistakes: a high recall can be achieved at the cost of an unacceptably high false positive rate. Accuracy, for
intrusion detection, due to the unbalanced nature of attack classes, is less informative, unless it is calculated for each class separately.

For outlier detection Parzen density estimator, k-means algorithm and one-class SVM were used by Giacinto and Roli (2008), and hidden Markov models, statistical methods and rule based learning by Han and Cho (2003). The results were mixed: Giacinto and Roli (2008) claimed that an ensemble model did not perform better on every subset of the data, whereas Han and Cho (2003) reported poor performance on the base classifiers, but much better once combined.

Another group of researchers started experimenting with ensemble models to increase classification accuracy by aggregating the results of multiple base classifiers. Papers written on ensemble modeling are Giacinto and Roli (2003, 2008), Han and Cho (2003) Chebrolu, Abraham and Thomas (2005), Folino, Pizzuti and Spezzano (2005), Mukkamala, Sung and Abraham (2005), Abadeh et al. (2007) and Tian, Liu and Xiang (2009). These works all used the KDD Cup 1999 dataset, the differences were the sampling methods and feature selection approaches applied.

Result combination was based on a simple function of predictions, such as simple majority vote, average vote, rule-based evaluation, etc. With the exception of Tian, Liu and Xiang (2009), no paper utilized a machine learning model for prediction synthesis in a stacking ensemble classifier.

Tian, Liu and Xiang (2009) created a distributed learning model using artificial neural networks in a two-staged approach: in the first stage, a network learns on a random subset of the KDD Cup 1999 10% dataset’s features. In the second phase, the class predictions of these models are collected by a final classifier improving prediction performance.

Chebrolu, Abraham and Thomas (2005) used a three-phased approach. They first performed feature selection on a sample created from the KDD Cup 1999 10% dataset (11 982 connection records divided randomly into of 5092 training and 6890 test). Training involved the creation of a Bayesian network and CART decision trees which were tested in their separate phase. Later the two models were combined in a weighted bagging classifier with improved detection performance.

Folino, Pizzuti and Spezzano (2005) used distributed parallel genetic programming to generate decision trees that provided accurate predictions. Ensemble modeling is
performed using simple majority vote between each generated decision tree. The solution was tested using the 10% sample of KDD Cup 1999 partitioned equally over the parallel environment of 10 nodes. Performance has been measured using confusion matrices from which many metrics could be calculated, including recall.

Almseidin et al. (2017) compared several models on the KDD Cup 1999 dataset, reporting random forest classifiers having the best overall performance in terms of precision, recall and AUC.

So-In et al. (2014) manually extended the dataset of the KDD Cup 1999 competition with a new class and synthetic data based on BOTNET signatures. Their comparison covered decision trees, sequential rules, artificial neural networks, naïve Bayes, k-nearest neighbor algorithms and support vector machines in four different experimental scenarios:

- Normal and attack traffic detection
- Attack classification
- Attack classification with new classes: U2R and the artificially generated BOTNET
- A detailed attack classification model with 8 classes (each having at least 900 observed network connection)

So-In et al. (2014) reported good accuracy (~90%). The use of accuracy in this case is justified as the authors carefully balanced out class distributions prior to commencing the experiments and used per class accuracy measures as well. The best performing models were decision trees, neural networks and k-nearest neighbors.

Petersen (2015) used the NSL-KDD dataset and four machine learning algorithms (ID3 and CART decision trees, k-nearest neighbor and naïve Bayes) to perform three hypothetical experiments with different class membership (2, 5 and 40 classes as target features). The author used accuracy, false positive rate and average cost. A secondary analysis evaluated feature importance using information gain as metric. The results shown that k-nearest neighbor and ID3 decision tree algorithms had the best performance, however the author failed to notice that the algorithms were sensitive to the choice of target class. Among the target classes, the 2-class models performed best, the author noted, however, that a case for a 5-class classification could be created, as it provides additional clues for the system administrators to act on.
Elhag et al. (2015) proposed a genetic fuzzy system for classification in One-on-One pairwise classification model. Their aim with using such a technique was to improve prediction performance on minority classes. Their method of choice was FARC-HD, which was initially based on association rule mining, but later has been extended for classification. Two important takeaway concepts in this paper were the difficulty of multiclass classification and the base rate fallacy. Multiclass classification can be transformed to a series of one on one binary classification questions with the results combined later using a score matrix. Base rate fallacy happens when a severely unbalanced data is evaluated using recall and false alarm rates. The recall for some intrusive connections becomes less than the false alarm rate, and therefore will be ignored in a final model. Elhag et al. (2015) used a 10% sample of the KDD Cup 1999 dataset. Based on Tavallaee et al. (2009), they removed duplicate observations, thus using 145,585 observations with slightly different class distributions than the original. In their experiments, the authors split the dataset to a 10% training, and to a 90% test sample with stratified split method, except for the U2R attack class, which was oversampled at 50%. Performance evaluation shown comparable results to other fuzzy rule generation algorithms and to decision trees. The proposed genetic fuzzy system performed well with underrepresented classes as well, while maintaining low false alarm rates.

Bouzida et al. (2004) experimented with k-nearest neighbor and decision tree approaches augmented by principal component analysis on the complete 10% KDD Cup 1999 dataset. They reported good classification performance on as low as four principal components for both models. Predictions of R2L and U2R classes, however, remained a challenge.

Yao et al. (2017) developed a new Hybrid Multi Level Data mining (HMLD) system for intrusion detection. It consists of three components:

- **MH-DE** (Multi-level Hybrid Data Engineering): tasked to perform data engineering tasks: convert categorical features to numerical and normalization. Also performs data preparation for one versus rest classification datasets and performs feature selection on the generated samples.

- **MH-ML** (Multi-level Hybrid Machine Learning): responsible for detection model training. First clustering is performed on each data sample then one model is
trained for each cluster from the selection of SVMs, ANNs, DTs and RFs. These models are not evaluated immediately.

- MEM (Micro Expert Modify): generates “impurity data” from the misclassified data (from MH-ML module). A new decision tree model is trained on the misclassified dataset to improve prediction performance.

HMLD achieved better classification performance using the KDD Cup 1999 10% sample for training and the complete KDD Cup 1999 labeled test data for testing, than many non-ensemble and ensemble approach used before, even on the more challenging minority classes.

At the end of their review, Tsai et al. (2009) proposed three possible future directions for research:

- Advocating the comparison of hybrid models and ensemble classifiers in terms of prediction accuracy.
- Combine hybrid and ensemble models.
- A comparison of potential feature selection algorithms for intrusion detection.

Bhuyan, Bhattacharyya and Kalita (2014) proposed the following issues, questions and research topics:

- The nature of attacks keeps changing over time, therefore adaptability of models is a must.
- A high rate of false alarms should be avoided (however, it cannot be eliminated completely).
- There is an overarching need for benchmark intrusion datasets.
- Developing a fast and appropriate feature selection for each attack class.
- Selection of non-correlated and unbiased classifiers for building an effective ensemble approach for anomaly detection.

Buczak and Guven (2015) advised the following criteria to compare machine learning algorithms with one another:

- Performance measures do not work for comparison, as the machine learning algorithms were trained and tested on different samples of the same dataset, making the results incomparable.
Due to the ever-changing cyber-attack types, intrusion detections need to adapt quickly. IDS model training is performed when traffic is the lowest, usually at night. It is expected form the training process to not take 24 hours. A low time to train therefore is a key element of algorithm evaluation.

Intrusions happen fast. A fast classification of an observed connection can improve reaction time and shows packet processing capability of the intrusion detection system.

To help administrators examine the model features and to patch the system more easily, a model with lower complexity is preferred.

Buczak and Guven (2015) furthermore gave the following advice on creating machine learning models for intrusion detection:

- Intrusion detection is a field with rapidly changing environment, models must be trained daily, or when a new intrusion type is discovered. For faster training, the whole model should not be trained again, but incrementally as the administrator feeds new data to it.
- The KDD Cup 1999 dataset, as a benchmark, is widely accepted and used, however it has its own flaws. It contains too many observations and the target class is unevenly distributed. Many try to combat both by sampling the dataset, which makes result comparison difficult. In this regard, the NSL-KDD dataset constructed by Tavallaee et al. (2009) is a better alternative.

Challenges from a big data perspective are provided by Cárdenas, Manadhata and Rajan (2013). The challenges are valid, as real-life intrusion data bears many characteristics of big data, even for moderately sized systems.

- Storing and retaining data is not economically feasible.
- Performing analytics and complex queries on large, structured data sets is inefficient.
- Traditional tools were not designed to analyze and manage unstructured data.
- Big data systems use cluster computing infrastructures. Distributed processing poses new challenges in algorithm design.

Dua and Du (2016) also identified multiple challenges for data mining algorithms in intrusion detection:
• Modeling large-scale networks. Creating graphs based on large networks is a difficult task.
• Discovery of threats. The sheer volume of heterogeneous data, the dynamically changing threats, and the severe imbalance between normal and anomalous classes complicates threat detection.
• Network dynamics and cyber-attacks. Novel data mining methods are necessary to predict future attacks by evolving malware and launch defenses.
• Online learning methods for dynamic modelling of network data. The most common way to evaluate dynamic data is to create a moving windowed aggregation. The main challenge here is to determine the window size.
• Modelling data with skewed class distribution to handle rare event detection. There is a fundamental asymmetry in anomaly detection problems, many normal activities and much fewer attacks. Standard machine learning algorithms have been found to be biased towards the majority class. Classification should be more focused on classifying the minority class as attack or anomaly.
• Feature extraction. One of the biggest challenges in anomaly detection is to select features that best characterize the user or the system usage patterns. This is often carried out to reduce the dimensionality of the data.

To summarize, the following areas need substantial attention:

• Design hybrid, signature and anomaly detection approaches and/or ensemble models for comprehensive, unbiased intrusion detections.
• Construct a model that is fast to adapt and fast to make predictions.
• Mind the data: if the KDD Cup 1999 dataset is used, then an appropriate sample, and a good feature set should be chosen. With NSL-KDD dataset, sampling is less of a challenge, but feature selection should be performed nonetheless.
• When measuring performance, false alarm rate and recall are more important than accuracy (however, false alarm rate cannot be eliminated completely). When comparing with works from the literature, always check the number of observations in a sample and the target class distribution.
• The research community experimented with many data mining algorithm. The best performers were decision trees, k-nearest neighbor methods and artificial

3. Research overview
This section has been assigned the task to provide an overview of the research I have conducted, demonstrated with the tools, techniques and considerations of the design science methodology and the CRISP-DM process. These two have many intersections, as some steps in the CRISP-DM process supports design science activities.

3.1. Research method
The methodology used for model design and creation follows a top-down pattern (Figure 21): as the goals outlined in this dissertation can be achieved by creating and experimenting with and algorithmic artifact, design science was found to be a fitting methodology for this kind of task. Furthermore, the artifact to be created has a lot of ties with data science: being a predictive model, some of the concepts and considerations of CRISP-DM methodology for creating data science models can be applied to it as well, forming the second pillar of abstraction. Finally, the methodology section will outline the exact process for ensemble model creation, with the necessary data preprocessing, training and testing steps involved, forming the lowest level of abstraction.

Figure 21: The methodological abstraction levels followed in this dissertation. Source: Own Edit.

3.2. Context
The context of intrusion detection, apart from the details discussed in section 2.1, were elaborated in Ahamad et al. (2009). They identified five reasons for developing intrusion detection systems:

- Threats from Malware: Hackers use malware programs to steal private information. They leverage the vulnerabilities of web site structures, social network systems and document transmissions that do not scan for malware. Once intrusion occurs, the malware will track the user’s keystrokes, spy on the users browsing habits and send the user’s personal information to the attacker.
• **Threats from Botnets:** Botnets are a group of machines that are hijacked and coordinated by attackers. Bots in a botnet report to a hidden master computer controlling the botnet. Computer and internet users suffer privacy breaches or financial losses, loss of valuable data, and damage to computer systems caused by botnets.

• **Threats from Cyber Warfare:** Cyber-attacks are critical military actions. The dependence of traditional infrastructure on cyberinfrastructure leaves many vulnerabilities for cyber warriors to exploit. Cyber defense is an inevitable, but challenging goal of military forces around the world. An efficient cyber defense requires collaboration between countries, states, institutions and industry members, because cyber-attacks can be launched from various locations and sources. Legislation lags the development of ICT technologies. Because of this, international cyber laws are lacking, making cyber defense an even bigger challenge.

• **Threats from Mobile Communication:** The development of mobile communication caused the proliferation of reliable services. Investigations have shown that even financial transactions appear in mobile services, which draws the interest of hackers as well. The mobile infrastructure and devices provide several opportunities to steal valuable information. Research institutions are developing new ways to protect against voice fraud and phishing.

• **Cyber Crimes:** The very first challenge in cyber-crime is to define what cyber-crime is. Different jurisdictions define cyber-crime differently as they correlate to local situations. Prospering e-commerce or online business entices cyber criminals, many purchase attack platforms to carry out their activities. This is usually done by exploiting vulnerabilities in the e-commerce or online services industries. Combating them is difficult as they do not leave any traces behind. Combating cyber-crimes requires effort in two perspectives: First uniform cyber laws need to be enacted. Second, advanced intrusion detection technology needs to be developed to defend against criminals.

All the above and more fuel the efforts aimed at creating new and better intrusion detection systems. The goals of actors in the social context can be summarized in the following points:
• Risk mitigation: reduce the chance of intrusion, information loss, or fines in the form of potential lawsuits. Reduce system downtime due to DDoS attacks.
• Infrastructure and national security: prevent the sabotage of key infrastructural elements, such as electricity and water supply, that are increasingly reliant on the information infrastructure.
• Protection of private information: restrict access to sensitive information, such as credit card numbers, bank account and personal information.
• Protection of government secrets: as an extension of the point above, all governmental bodies store information, that were not meant for the public eye. These information assets must be protected as well.

The knowledge context has already been discussed in section 2 for both the intrusion detection and machine learning backgrounds of the dissertation. Out of all, there are two knowledge context elements that deserve mentioning: neural networks, which provide the algorithm to learn intrusion patterns, and CRISP-DM (Figure 3), which forms the methodological framework to conduct experiments with.

3.3. Research goals

The goal and design problem of this dissertation is to provide a novel intrusion detection solution applying machine-learning methods. Accordingly, the two research aims to achieve:

1. To create an intrusion detection model that can outperform the ones introduced in contemporary scientific literature, measured by detection performance indicators. Performance in this context can be described as the portion of attacks correctly and/or incorrectly classified as being part of normal activity and vice versa.

2. To identify methods that can improve performance on complex and/or rare event detection problems. The task of intrusion detection fits these problems, as the data available is heavily skewed towards the more common normal, rather than the rarer malicious activity. Some of these methods involve synthetic sampling to feed more balanced training data for the model, hyperparameter optimization to find the overall best performing setup for the machine learning model, and ensemble methodologies, to create composite models for improved predictions.

Based on these two research goals the research questions can be formulated.
3.4. Research questions

1. Is machine learning a suitable approach for intrusion detection? If machine learning is a proper technique for intrusion detection, which are the appropriate models?

Finding the right machine learning model is a challenging task. It is affected by the selected intrusion detection method (signature detection or anomaly detection) as well as the available dataset and the sampling method chosen for that dataset.

The most common and best working non-ensemble machine learning algorithms in intrusion detection are decision trees, artificial neural networks and k-nearest neighbor algorithms for signature detection, and k-means algorithms and statistical methods for anomaly detection. Each algorithm has difficulties though:

- Decision trees are prone to overfitting, unstable (a slight change in data causes entirely different decision trees) and perform poorly on unevenly distributed training classes.
- Artificial neural networks, like decision trees, are prone to overfitting, and generally have long training times.
- K-nearest neighbor algorithms are fast, but need all data for accurate predictions, therefore they scale poorly on data that has a lot of observations.
- Anomaly detection itself is a less researched area.

Countless works in the literature have proven that a good combination of machine learning algorithms can detect intrusions at a high rate with low amount of false alarms. As the experiments conducted in this dissertation will show.

2. Which type of intrusion detection method is more effective from the following ones: misuse detection by classification, anomaly detection by outlier analysis or a combination?

This is one of the more recent challenges of intrusion detection research, also highlighted by Dua and Du (2016). On one hand, signature detection has high recall and low false positive rate, are easy to implement, and provide detections quickly. However, they are incapable of detecting new, unknown attacks. On the other hand, anomaly detection aims at building a profile of normal attacks, and then detect anomalous (or attack) traffic based on the difference from this normal profile. Anomaly detection can capture unknown
attacks; however, it is difficult to make a difference between attacks and anomalous traffic, as the latter might include normal connections as well as highlighted in Ippoliti (2011, 2013). Therefore, anomaly detection will have high false positive rates. In a good intrusion detector, both recall and false positive rate are acceptable. Signature and anomaly detectors use compensatory functions and techniques, so it might be a good idea to combine them into new hybrid detectors.

A simple combination of the two techniques is not enough, a more purposeful approach must be followed. One must decide two questions:

- Find the best candidate algorithms for the individual signature and anomaly detectors.
- Find the optimal way to integrate the detection algorithms given in the previous step to achieve the best balance of recall and false positive rate.

Based on Zhang and Zulkernine (2006), Zhang, Zulkernine and Haque (2008), Dua and Du (2016) proposed four integration approaches (Figure 22):

- **Anomaly-Signature sequence detection**: are designed to reduce false positive rates by excluding suspicious patterns that are not classified as alarms by the signature detector (Figure 22.a).
- **Signature-Anomaly sequence detection**: are designed to improve the detection of unknown attacks missed by the signature detector (Figure 22.b).
- **Parallel detection**: are used to correlate misuse and anomaly detection results to provide a stronger detection decision (Figure 22.c).
- **Complex mixture detection**: any detection approach using anomaly and signature detectors, that did not fit in the categories above.
To achieve better detection performance, therefore one must use anomaly and signature detection approaches in a combined manner, not just one or the other. The literature predominantly analyzed signature detection and relatively less anomaly and hybrid detection, leaving much unexplored territory for hybrid intrusion detection.

Effectiveness in this sense is not really about the training or testing time the models take, but more along the lines of model performance measured by standard metrics derived from the confusion matrix: accuracy, precision, recall, $F_1$-score are all good measurements of predictive performance. Here, due to the special characteristics of intrusion detection, recall is a favorable metric compared to accuracy.

3. What is the level of model performance that can be expected in an intrusion detection task?

Based on the literature review, contemporary intrusion detection literature is facing the following challenges:

- Predominant use of the accuracy measure for performance on data with unevenly distributed classes.
- Create their own sample of the available dataset, making performance comparisons between articles difficult if not impossible.
- Focus mostly on signature detection, less on other techniques.
- Detecting the underrepresented classes is one of the most challenging tasks, because even a good recall dominates the false positive rate.
There is a high variation on possible model performance measurements. Therefore, a criteria system has been set up for selecting works from the literature for comparison with the models created to test the assumptions of this dissertation.

- Emphasis on recall / detection rate: though, accuracy is the most common metric in use, it is inappropriate for performing detections on imbalanced data. Better metrics are ROC curves and AUC, which is only applicable for binary detection problems, or recall. Literature with recall as the target model performance indicator are favored to those with accuracy, though three important factors are taken into consideration: due to its commonality, accuracy cannot be ignored completely, and that a tradeoff between recall and precision exists, following recall values blindly should be avoided. Third, it is already a complicated task to find good intrusion detection papers that refer to recall directly by its name. Other works claim to have used detection rate, when, the definition fits accuracy instead.

- Sampling is the second source of complexity and result variance. Different samples result in different models with different performance measurements. Instead a new testing setup is proposed where a model could be trained on the provided training set or a sample from the same dataset, but model validation is only performed on the complete test set. The test set is only transformed to accommodate the format requirements set by the model using transformers set up on the training set only.

- Compare models with models of the same kind. Simple classifier models should not be compared with ensemble-based models. Similarly, results from a signature detector should not be compared to results from a hybrid model.

- Techniques like cross-validation can provide a broad understanding of model performance. This study strives to apply these techniques where possible.

These requirements worked as filters determining which works from the literature were used in the comparisons.

### 3.5. Experiment Design

The experimental designs were carried out using the CRISP-DM framework, commonly used for data science modeling tasks and previously shown on Figure 3. This model fits the design science methodology, as both describe highly iterative processes with numerous feedbacks and recursions to previous steps. Both involve rigorous design,
implementation and testing procedures, and the evaluation of social and knowledge context.

Though, the two methodologies have minor differences too. One of the key differences are the goals of the two. The main goal of design science is not only to deliver a designed working artifact, but also to answer scientific questions aimed at the artifact, at the context or at the relationship between the two. Compared to this the goal of CRISP-DM is more practical. It is interested in delivering a machine learning algorithm preferably as the part of a working business solution delivering value to both customers and the organization itself. The CRISP-DM approach therefore is more focused on evaluating the business context and the effects on the business context, rather than on answering research questions.

Some personalization of the CRISP-DM process model hence, will be necessary. These adjustments are not only permitted, but also encouraged by the designers of CRISP-DM, as they intended it to be a collection of best practices within the field of data science, rather than a rigid standard. Some of the changes compared to the CRISP-DM process model are:

- Greater emphasis on the wider context of the intrusion detection model: This includes both the social scientific context via literature reviews and the knowledge context by covering the data scientific tools and techniques in use. Both were already covered in section 2 of this dissertation report.

- More emphasis on model evaluation involving the comparison of model performances on three levels:
  - Where the model is designed as an ensemble, the base classifiers are compared to one another.
  - Comparisons are also conducted between the different iterations of the intrusion detection model delivered as part of the design process. This supports the disclosure rule of the design science process as well.
  - Finally, the best performing iteration is compared to other works available in the scientific community, with a higher level of emphasis on detection rate.
The deployment process is not covered. The primary goal of design science is not to use the designed artifact, but to answer scientific questions which may or may not involve the use of the delivered artifact in context. In this dissertation no question asks how the context changes with the deployment of the model.

Following the CRISP-DM approach, the next step after acquiring business understanding (in this case, gathering information about the social and knowledge contexts) is data understanding. After reviewing the literature, particularly J. Stolfo et al. (2000), McHugh (2000) and Tavallaee et al. (2009) the most common datasets for intrusion detection in use are:

- DARPA 1998 & DARPA 1999
- KDD Cup 1999
- NSL-KDD

These datasets are all the products of an experiment conducted in 1998 by MIT Lincoln lab to survey the state of the art in intrusion at the time. During the experiment about 5 million records were collected in 5 weeks in the form of raw tcpdump logs. The data simulated the traffic of a typical Air Force LAN, while the researchers carried out multiple network attacks against it.

The most popular out of the proposed is the KDD Cup 1999 dataset. It consists of ~5 million network connection records for training, and another ~3 million unlabeled records for testing intrusion detection models. The altogether ~8 million records might be too difficult for an intrusion detection system to handle; therefore, the authors of the KDD Cup 1999 dataset provided a 10% stratified sample of both the attack and test datasets, the latter also having an attack labelled version. The total number of features available is 41. Designed primarily for signature detection, one target variable contains numerous different attack types each belonging to five distinct attack categories:

- **DoS**: denial of service attacks aimed at disabling crucial systems or system components.
- **R2L**: unauthorized access from a remote machine.
- **U2R**: unauthorized access to local superuser (~admin) privileges by a local unprivileged user.
PROBE: surveillance and probing, often not the attack itself, but could be an indication of a future attack.

And NORMAL legitimate behavior.

Compared to the previous DARPA 98 and 99 datasets, the features of KDD Cup 1999 are better organized and described, and, as part of some data preprocessing new derived features were created as well by J. Stolfo et al. (2000). These features can be grouped into four categories:

- **Intrinsic features**: features describing all network connections, regardless of user intentions.

- **Content features**: capturing information on the content of each network connection.

- **Time-based traffic features**: features aggregating the connections that had the same destination host or service as the connection in question in the past 2 seconds.

- **Host-based traffic features**: as a counterpart to time-based traffic features, host-based traffic features were created to capture aggregate data not over the past 2 seconds, but over the previous 100 connections.

Following its inception, some researchers, such as McHugh (2000) and Tavallaee et al. (2009) expressed the following criticism with the KDD Cup 1999 dataset:

- The classes were developed from the attacker’s point of view, which is not what a detection system is looking for in a realistic environment.

- Object of Study questions: In the predecessors of KDD Cup 1999, it was unclear whether the researchers used connection packets or connection sessions as the basis of their analyses, making it difficult to interpret the results.
There are issues with the class distribution as well. Figure 23 shows class distribution for the 10% sample of the KDD Cup 1999 dataset. There is too much DoS attack, and not enough normal traffic. This distribution is not realistic, a real-life environment has a ratio closer to 98-95% to 2-5% between normal and any kind of attack traffic. Nonetheless, the challenge of class imbalance persists, the only factor that changed is the class that is in imbalance. Such inequality in class distribution can be corrected by employing one of multiple strategies recommended by Brownlee (2015):

- Collect more data. As the KDD Cup 1999 has concluded long time ago, this alternative is improbable.
- Change the performance metric from accuracy to something different, discussed in section 2.2.
- Resampling the dataset: one can use oversampling on the less represented classes, and under sampling on the better represented ones. This serves no benefit to the experiments by itself, as the absolute minority class has <100 observations. Therefore, even if the minority classes are 100% oversampled, their number is insufficient when put into comparison with the majority classes. And when the majority class is balanced out, then the size of the training sample would be too small for any meaningful model to be trained. However, sample balancing could potentially work as part of a more robust sampling procedure.
Generate synthetic samples: The idea behind synthetic sampling is to generate samples where the records are not necessarily from the original dataset but were created randomly on the fly. Two ways of generating synthetic samples are reversed Naïve Bayes algorithm, or, when non-linear relationships are important as well, SMOTE. This approach could potentially work as the second leg of the process aimed at creating a balanced training sample.

Try different algorithms: use not just one data mining algorithm on a dataset but try out more and see which ones work best. This is the thought process behind the application of model ensembles.

Use penalized models: Penalized classification imposes an additional cost factor to misclassification during training.

Use a different perspective: view the dataset from the perspective of the area studying it. This usually involves different machine learning algorithms, for example, clustering or outlier analysis instead of classification. This is the idea behind anomaly and hybrid detection models.

Tavallaee et al. (2009) also identified an issue similar to class imbalance with the KDD Cup 1999 dataset: a large number of redundant observations (Table 3 and Table 4). About 75% of the test set and 78% of the labelled training set is duplicated. This redundancy often caused in research papers prior to 2009 to have biased intrusion detectors towards the duplicate records. To alleviate this issue, Tavallaee et al. (2009) proposed the new NSL-KDD dataset. It comes in two versions: one with binary labels and the other with 5-class labels, both with their respective training and test sets.

<table>
<thead>
<tr>
<th></th>
<th>Original Records</th>
<th>Distinct Records</th>
<th>Reduction Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attacks</td>
<td>3,925,650</td>
<td>262,178</td>
<td>93.32%</td>
</tr>
<tr>
<td>Normal</td>
<td>972,781</td>
<td>812,814</td>
<td>16.44%</td>
</tr>
<tr>
<td>Total</td>
<td>4,898,431</td>
<td>1,074,992</td>
<td>78.05%</td>
</tr>
</tbody>
</table>

Table 3: Statistics of redundant records in the KDD train set. Source: Tavallaee et al. (2009)
Despite the numerous criticisms formulated, a large portion of literature still uses the KDD Cup 1999 dataset. Therefore, the decision has been reached to prefer this dataset over NSL-KDD, as a much wider array of research papers were available for comparison this way. The other benefit is the use of a dataset that has been evaluated many times before, and now works as a benchmark for intrusion detection models.

The object of study of this dissertation are intrusion detection models created by combining machine learning algorithms in the form of an ensemble. The design and implementation of one, however, is not an easy task due to the specifics of the dataset. To find an appropriate model, one must perform multiple iterations. Further elements of the CRISP-DM process are discussed in terms of these iteration, where each produced a new refined version of an intrusion detector. Figure 24 shows the detection models created as part of this iterative process:

- **Version 0** (Prototype): the first prototype of the model is outlined and evaluated in Brunner (2017). It involved the creation of a Decision Tree bagging classifier trained on a MapReduce-like architecture.

- **Version 1** (Neural Network Stacking Ensemble): Involved the creation of a stacking ensemble from neural networks trained on different features of the same dataset. Performance improvements were achieved with the use of a more robust sampling process and of hyperparameter optimization.

- **Version 2** (Migration to TensorFlow): The neural network ensemble has been moved over to TensorFlow + KERAS platform achieving faster training time, with the added benefit of GPU utilization. Further improvements in training time and prediction performance are expected by using gaussian processes for hyperparameter optimization. This iteration, at the time of writing this dissertation plan is still in its early stages, therefore evaluation results are not available.
Further sections show how the remaining steps of CRISP-DM were implemented throughout the different versions of the intrusion detector.

### 3.5.1. The decision tree bagging model

The first machine learning model used for intrusion detection was built using decision trees organized into a bagging ensemble on a parallel map-reduce environment. This environment is discussed in detail in Brunner (2017). The programming language of choice was Java, using the WEKA API for machine learning.

**Data preparation**

![Diagram of data preparation](image)

The steps of data preparation are outlined on Figure 25. First, the unique attacks had to be organized into their respective categories. To achieve the desired outcome a conceptual hierarchy was used, available in Appendix A.
Next, stratified sampling was performed on the 10% sample of the KDD Cup 1999 training dataset. Altogether 4 different datasets with alternating target features were created along the following criteria:

- Target variable kind: two- and five-class classification as described above.
- Sample size: small or large. This had implications on training time, reduced by parallelization. The size of the large sample is exactly double of the small sample.
- Purpose of dataset: test or training. Due to frequent memory overflow errors of the Java platform, the training and test datasets had to be switched around for two-class and five-class classification tasks. Table 5 shows the exact sampling setup.

<table>
<thead>
<tr>
<th>Target variable</th>
<th>Training</th>
<th>Test</th>
<th>Sample size</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 classes</td>
<td>3,000</td>
<td>5,000</td>
<td>S</td>
</tr>
<tr>
<td>2 classes</td>
<td>5,000</td>
<td>3,000</td>
<td></td>
</tr>
<tr>
<td>5 classes</td>
<td>6,000</td>
<td>10,000</td>
<td>L</td>
</tr>
<tr>
<td>2 classes</td>
<td>10,000</td>
<td>6,000</td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Sampling setup of the prototype intrusion detector. Source: Brunner (2017)

Apart from the target recategorization and data sampling, no feature selection or feature grouping was performed, and the data remained non-normalized.

**Modeling**

The modeling was set up to work in a parallel map-reduce environment in three different architectures:

- 1 processor, 2 cores: the benchmark architecture
- 1 processor, 4 cores
- 2 processor, 8 cores

Out of all the cores, the first one was reserved for a master thread tasked to distribute stratified subsamples to all the remaining cores to train decision tree models (Figure 26). Each subsequent core then produced and sent back their predictions to the master, where the final prediction is generated as a simple majority vote between the workers.
To mitigate variation arising from random chance, training was repeated three times for the combination of each sample size (small or large), architecture setup (1 processor 4 cores, 2 processors, 8 cores) and target feature kind (two- and five-class prediction). Altogether, including the benchmarks, the training and testing process was repeated 28 times.

**Evaluation**

Data measurement and data collection were determined by the target feature. By default, accuracy, precision, recall and F1-score measurements were collected throughout the executions, the latter three have been macro-averaged for convenience. These were expanded with ROC AUC for two-class classification. Due to a lack of knowledge on ensembles, only base classifier performances were collected, aggregate performance was not. Moreover, in some cases the base classifiers were unable to detect specific attack types, thus precision, recall and F1-score would have been impossible to calculate. A standard behavior in many programming environments in such cases to set the corresponding metrics to 0 for the appropriate class, then calculate the micro or macro averaged measures. Thus, the experiment followed this behavior in a manual post-processing step, as the WEKA API did not support it.

Due to the parallelization on the map-reduce architecture, the execution time was measured as well, although following sections in this dissertation dropped model training and testing time evaluations.
Points of improvement

This first version is best described as a version 0, or as a prototype intrusion detector. It had many flaws:

- Only the individual classification performance for each decision tree were measured, the aggregate performance could only be evaluated as the mean of each base classifier, which does not reflect the real classification capabilities of a bagging ensemble.

- Java and the WEKA API, though useful on their own right, have counterparts that are better at performing data preprocessing, model training and testing tasks. Two examples are python and R programming languages. Both are less prone to malfunction and easier to maintain. In addition, python has readability advantages over most programming languages as well.

- More robust sampling methods are to be explored, having a key effect on model performance.

- New machine learning models were recommended for use in detection models, particularly artificial neural networks.

- Opportunities related to feature selection and feature group creation were left untouched.

- Out of two- and five-class classification, only the latter should be kept, being the more interesting and challenging task. This is supported by the nature of network intrusions as well, after all, different mitigation policies should be applied to DoS attacks than to R2L or U2R attacks.

However, some findings of this early version are undeniably valuable. For example, the application of model ensembles was a forward-looking idea. With all the above considered, a new artifact was designed.

3.5.2. The stacked neural network model

The next intrusion detection model has been implemented using the Python scientific stack (a collection of Python modules designed for data manipulation and data scientific tasks, the core modules being pandas + numpy + matplotlib + scikit-learn) in Brunner,
Kő and Fodor (2019, submitted work). A new stacking ensemble of artificial neural networks was created and evaluated for intrusion detection performance.

Data Preparation

Figure 27: Data preprocessing for the neural network stacking detector. Source: Own Edit.

Figure 27 shows the modified data preprocessing performed on the 10% sample of the KDD Cup 1999 dataset still. The new steps added are highlighted in green, although the previous two were altered as well:

- Categorical features were one-hot encoded to be more appropriate for processing by the neural networks.

- The target variable was created using the same conceptual hierarchy as in 3.5.1., but only for five-class classification.

- A simple feature selection was performed to remove features with no variance among observations (~no information content). This feature selection was based on the relative deviation metric.

- Sampling has been fundamentally changed to generate balanced samples in two steps. Step one performed balancing stratified split, where underrepresented classes had higher probability to be selected in the sample. The exact sampling fractions of this step are visible in Table 6. The second step balanced the sample further by performing SVM-SMOTE synthetic sampling on the intermediate sample from the step prior. With this two-step approach a 100% balanced sample could be created. One might ask whether this sample is a valid representation of the original dataset. This has been validated in a separate experiment where the sampling process has been generated 150 times, then compared to the 10% sample of the KDD Cup 1999 training dataset. This evaluation of the sampling strategy was based on the nonparametric two-sample Kolmogorov-Smirnov test from
statistics. The null hypothesis of the K-S test states that the two samples were drawn from the same statistical distribution. These K-S tests were performed for each class, feature and iteration up until the acceptance or rejection of the null hypothesis. The rejection and acceptance are then aggregated per class to provide an idea on how well each sample matches the original.

<table>
<thead>
<tr>
<th>Sampling fraction</th>
<th>Normal</th>
<th>DoS</th>
<th>Probe</th>
<th>R2L</th>
<th>U2R</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.50%</td>
<td>0.50%</td>
<td>50%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 6: Sample fractions to balance class distributions before SVMSMOTE. Source: Own Edit.

- Feature group splitting was performed according to the findings of J. Stolfo et al. (2000). The sample was grouped into intrinsic, time-based traffic, host-based traffic and content feature groups.

- Finally, in the last step of data preprocessing, the training sample was normalized. This is a requirement to train neural networks.

**Modeling**

Multiple neural networks were trained, one for each feature group and one to act as the final aggregator model. The experiment setup is visible on Figure 28.

![Diagram of model creation and prediction process](image)

Figure 28: The model creation and prediction process of the stacked neural network model. Source: Own edit

An important element to stacking ensembles is the variance of the base models, which is usually achieved with different kinds of models. In this design, the variation was rather achieved with the training data fed to the models.
Every base model and the aggregator model were trained in a similar process. First, each were optimized with grid search hyperparameter optimization supported by five-fold cross validation for better stability. The target metric for optimization was recall and the hyperparameters changed were the learning rate, the learning rate decay and the momentum. The number of hidden layers, as well as the number of neurons per hidden layer were fixed, as grid search would have taken too long to execute. Once the base models were trained, the same process was performed for the aggregator model, with base classifier probability predictions as input data. This way, the aggregator model learned to refine the predictions of the base models, providing improved results.

**Evaluation**

For evaluation, the standard collection of accuracy, precision, recall and F1-score measures were used, obtained from testing the ensemble with test data of the KDD Cup 1999 dataset. The test dataset has been transformed using the one-hot encodings, feature group splits and normalizers created using the training set to avoid the problem of information leakage.

**Points of improvement**

This experiment has taken a major step forward in terms of quality and classification performance compared to the prototype. However, a few new issues were identified:

- Although not measured with research intent, the training process with hyperparameter optimization took a significant amount of time, which was a result of several factors: the notoriously long training time of neural networks, the grid search algorithm (evaluates all parameter settings provided) and cross validation. TensorFlow with KERAS and GPU capabilities is a good candidate to improve this training time, with the likely benefit of improving model performance further.

- Hyperparameter optimization was carried out in the grid search setup, but randomized search was considered at some point as well. Both have flaws, grid search takes a lot of time (see above), while randomized search is not guaranteed to find the optimal performance. Gaussian hyperparameter optimization mixes the benefits (fast with few combinations tested and the returned result is sufficiently close to the global optimum) of the two search strategies. Moreover, it can search
in much larger parameter spaces, therefore more parameter features could be evaluated.

### 3.5.3. Neural networks on TensorFlow and KERAS

The next experiment is a natural evolution of the neural network stacking model. It has been created by implementing two major changes to the original training process: first, the platform of choice was changed from scikit-learn to KERAS supported by TensorFlow. Second, the training process now uses gaussian process instead of grid search for hyperparameter optimization.

According to Google Brain Team (2015) “TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications”. Developed by Google, it has quickly obtained popularity in the fields of machine learning and AI research.

The latest release of TensorFlow has native support for KERAS as a high-level API. According to the KERAS documentation Chollet (2015): “Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation”. KERAS allows for fast model implementation for research purposes.

**Data Preparation**

The data preparation step, as the changes focused on the model creation phase, remains largely unaffected compared to the previous version.

**Modeling**

The modeling phase saw the changes to the neural networks and the new optimization strategy (Figure 29).
The setup is the same as in the previous experiment: each base neural network is trained and optimized in a five-fold cross validation scheme, then the aggregator is trained on the base probability predictions. The difference is in the platform and the used optimization strategy. One of the many reasons for choosing TensorFlow was a better access to computational resources, namely the possibility to use the GPU of the machine the training is performed on. A question is the degree of benefit from doing so.

Furthermore, the gaussian optimization allows training to explore a wider range of optimization parameters, for example, the number of hidden layers, the number of neurons per hidden layer and the activation function together with the already explored learning rate, learning rate decay and momentum. This expanded optimization has the potential to find better optimal performance for the neural networks. The target measure for optimization remains recall, although a case could be made for average loss as well.

**Evaluation**

The evaluation is planned to be the same as it was in the previous experiment. The measures proposed for evaluation are accuracy, precision, recall and F1-score, with the potential inclusion of average loss. Additionally, training time could be measured between executions using / not using GPU resources. Thus far the following factors were identified with potential interest:

**Figure 29:** The model creation and prediction process of the stacked neural network model backed by TensorFlow and KERAS. Source: Own edit
• Faster training
  o TensorFlow + KERAS: the switch to a more modern API for developing neural networks also brings the potential in better optimized algorithms.
  o GPU Support: is the prime example of the above.
  o Gaussian optimization: just like randomized search, gaussian optimization test a limited number of parameter combinations.

• Model performance
  o TensorFlow + KERAS: Is there a difference between Multilayer Perceptron Classifiers in scikit-learn and KERAS models to the advantage of the latter?
  o Gaussian optimization: just like grid search, gaussian process can provide an optimal model with respect to the target measure by making a tradeoff between exploration and exploitation.

Points of improvement

Note that at the time of writing this dissertation plan, this experiment is still in its early planning stage, therefore some characteristics of it are still subject to change. Although, there still exists a king of intrusion detection solution capable of providing better predictions, namely hybrid detectors. At some point a new hybrid detector architecture is planned to be developed based on encoder networks.

4. Results

This section has been tasked with the evaluation of models results. It is organized similarly to 3.5: each model is discussed in order of their creation, starting with the decision trees bagging classifier, followed by the neural network stacking ensemble. Later, the results of the two are joined together and compared to external experiments of other researchers to see how well they perform against works available in the literature. The optimizations imposed by TensorFlow + KERAS and gaussian optimization will be discussed here as well, once results are available.
4.1. Decision tree bagging results

The results of the experiments can be seen in Table 7 and Table 8. The used metrics were accuracy, precision, recall, F1-score and, in the case of two-class classification, AUC. The original article in Brunner (2017) evaluated the effects of parallelization on classification performance. It has been concluded that parallelization had no effect on the performance of the machine learning algorithms. Moreover, sample size should not be a differentiating factor among performances either. Both factors were left out of the aggregated results calculation. The original measurement tables are available in Appendix B.

Two-class classification achieved moderate accuracy and AUC, high precision and low recall and F1-score. Compare this with five-class classification, where accuracy was high with the remaining measurements showing poor performance. The high accuracy in this case is misleading, as training and test samples were not balanced at the time, and accuracy tends to rate models better when they categorize most observations to the majority class.

<table>
<thead>
<tr>
<th>Measurement</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.788 (±0.0132)</td>
</tr>
<tr>
<td>Precision</td>
<td>0.925 (±0.0372)</td>
</tr>
<tr>
<td>Recall</td>
<td>0.513 (±0.0332)</td>
</tr>
<tr>
<td>F1-score</td>
<td>0.659 (±0.0266)</td>
</tr>
<tr>
<td>AUC</td>
<td>0.773 (±0.0237)</td>
</tr>
</tbody>
</table>

Table 7: aggregated measurement levels for the two-class classification. Based on Brunner (2017)

Two-class classification achieved the better results. The cause is that the models are only required to make a difference between normal and attack traffic (Petersen (2015)). Thus, the mistakes made between attack types remain hidden. Depending on the intrusion detection policies of the sponsor this may be unacceptable, as different controls need to be applied for different attack patterns. Therefore, the study of five-class intrusion detection performance is more favorable in future research, for example, with neural network stacking models.

<table>
<thead>
<tr>
<th>Measurement</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.979 (±0.0124)</td>
</tr>
<tr>
<td>Precision</td>
<td>0.491 (±0.0944)</td>
</tr>
<tr>
<td>Recall</td>
<td>0.458 (±0.0582)</td>
</tr>
<tr>
<td>F1-score</td>
<td>0.473 (±0.0708)</td>
</tr>
</tbody>
</table>

Table 8: aggregated measurement levels for the five-class classification. Based on Brunner (2017)
4.2. Stacking neural network results

The results obtained from the neural network stacking ensemble tested sampling stability and model performance. Model performance is further analyzed between base and composite levels in terms of the collected performance metrics.

Sampling stability

Sampling stability was designed to test the specialized sampling procedure. The process has been repeated 150 times. Kolmogorov-Smirnov tests were used at each iteration to compare the original 10% KDD Cup 1999 dataset and the generated sample per class and per feature, including one-hot encoded categorical columns. The decision was set to 0 if the null hypothesis of K-S test could not be rejected 1 otherwise. The result is an aggregation of these decisions down to target class levels. Table 9 shows the resulting table, containing mean rejection rate with 95% confidence interval.

<table>
<thead>
<tr>
<th>Ho</th>
<th>Normal</th>
<th>DoS</th>
<th>Probe</th>
<th>R2L</th>
<th>U2R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rejected %</td>
<td>0.16%±0.13%</td>
<td>0.02%±0.04</td>
<td>0.00%±0.00%</td>
<td>0.00%±0.00%</td>
<td>2.73%±0.76%</td>
</tr>
</tbody>
</table>

Table 9: Sampling validation results. Source: Own edit

Sample testing results show that probe and R2L classes match the original data for all explanatory features, while DoS and normal categories are matching their respective distributions. The only exception is U2R class where 2.75% (with a confidence interval of ±0.76%) of tests rejected the possibility that the sample has been drawn from the same distribution as the population (the 10% KDD Cup 1999 dataset).

Model performance

Each model (the four base classifiers and the aggregator model) were trained using the preprocessed training sample and grid search hyperparameter optimization. In total, 540 different combinations were tested, the exact details are visible in Table 10. The optimization searched for the initial learning rate, exponent for the decaying learning rate and momentum, for each of the models. More options were available, for example hidden layer counts and neuron per hidden layer counts could have been tested. In that case, however grid search would have taken too much time to find an optimized model. A compromise was to feed a set architecture for each of the models. The base classifiers were trained on a (40, 20, 10) architecture with three hidden layers, whereas the aggregator trained on a (10, 5) neural network. The original number of features plays no
role, the aggregator always receives \( n \times 5 \) inputs (where \( n \) is the number of base models), thus works with a more limited set of input features. The target measurement for optimization was recall, the evaluation was further improved by the application of a five-fold cross validation loop.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Base models</th>
<th>Aggregator model</th>
</tr>
</thead>
<tbody>
<tr>
<td>hidden layer</td>
<td>(40, 20, 10)</td>
<td>(10, 5)</td>
</tr>
<tr>
<td>activation</td>
<td>RELU</td>
<td></td>
</tr>
<tr>
<td>solver</td>
<td>Adam</td>
<td></td>
</tr>
<tr>
<td>alpha (L2 regularization parameter)</td>
<td>0.0001</td>
<td></td>
</tr>
<tr>
<td>learning rate type</td>
<td>inverse scaling</td>
<td></td>
</tr>
<tr>
<td>initial learning rate</td>
<td>0.01, 0.03, 0.05, 0.07, 0.1, 0.15, 0.3, 0.5, 0.7, 0.9</td>
<td></td>
</tr>
<tr>
<td>LR decay power</td>
<td>0.25, 0.5, 0.6, 0.7, 0.8, 0.9</td>
<td></td>
</tr>
<tr>
<td>momentum</td>
<td>0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9</td>
<td></td>
</tr>
</tbody>
</table>

Table 10: Hyperparameter settings, Source: Own edit

Testing concluded using the test set of KDD Cup 1999 using the transformers and models trained on the training samples. The results of this testing can be viewed on Table 11 and Table 12. Table 11 shows achieved model accuracies per class and trained classifier. The best base models were intrinsic and host-traffic for normal, host-traffic for DoS, time- and host-traffic for probe and R2L and host-traffic and content for U2R attacks. Overall, content model was the worst performing, however, it cannot be said that it is redundant, as it contains useful information on U2R attacks, partially confirming the findings of J. Stolfo et al. (2000). Finally, the aggregator model improved on the accuracy of all the base models.

<table>
<thead>
<tr>
<th></th>
<th>Intrinsic</th>
<th>Time-traffic</th>
<th>Host-traffic</th>
<th>Content</th>
<th>Aggregator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>91.76%</td>
<td>79.54%</td>
<td>92.10%</td>
<td>63.26%</td>
<td>92.13%</td>
</tr>
<tr>
<td>DoS</td>
<td>83.56%</td>
<td>85.11%</td>
<td>96.71%</td>
<td>17.16%</td>
<td>96.74%</td>
</tr>
<tr>
<td>Probe</td>
<td>85.27%</td>
<td>99.05%</td>
<td>99.16%</td>
<td>15.86%</td>
<td>99.07%</td>
</tr>
<tr>
<td>R2L</td>
<td>93.83%</td>
<td>93.36%</td>
<td>94.17%</td>
<td>74.41%</td>
<td>94.70%</td>
</tr>
<tr>
<td>U2R</td>
<td>98.62%</td>
<td>86.50%</td>
<td>99.32%</td>
<td>99.77%</td>
<td>99.92%</td>
</tr>
<tr>
<td>Overall</td>
<td>78.77%</td>
<td>74.94%</td>
<td>91.03%</td>
<td>15.31%</td>
<td>91.52%</td>
</tr>
</tbody>
</table>

Table 11: Stacking model accuracy with base model accuracies. Source: Own edit
Table 12 shows the remaining classification measures for each of the models. Recall, precision and F1-scores were all macro-averaged to get the results. Again, the aggregator model outperformed the base model except for recall on the intrinsic model. However, only the aggregator model could achieve high recall simultaneously with high precision scores showing its capability for better generalizations.

<table>
<thead>
<tr>
<th>Model</th>
<th>Intrinsic</th>
<th>Time-traffic</th>
<th>Host-traffic</th>
<th>Content</th>
<th>Aggregator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall</td>
<td>0.668</td>
<td>0.635</td>
<td>0.595</td>
<td>0.470</td>
<td>0.665</td>
</tr>
<tr>
<td>Precision</td>
<td>0.476</td>
<td>0.447</td>
<td>0.555</td>
<td>0.333</td>
<td>0.626</td>
</tr>
<tr>
<td>F1 score</td>
<td>0.402</td>
<td>0.442</td>
<td>0.525</td>
<td>0.269</td>
<td>0.582</td>
</tr>
</tbody>
</table>

Table 12: Macro-averaged precision, recall and F1-score of the stacking model. Source: Own edit

A more detailed per-class version of Table 12 is available in Appendix C.

**4.3. External comparison**

Additionally, to comparisons between base and aggregate classifiers, the aggregate classifier results are compared to intrusion detectors presented in contemporary scientific literature. The process of finding appropriate literature was complex. Most only published achieved accuracy, which has its own flaws as a metric of prediction performance. Some have not even published per class accuracy, which made finding appropriate studies more difficult. The other factor that made comparisons challenging is whether the authors of the literature in question applied any form of sampling or not, and where did the training and test datasets come from. Considering these difficulties the works of Chebrolu, Abraham and Thomas (2005), Tian, Liu and Xiang (2009) were selected for comparison. The details are available in Table 13.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>92.13%</td>
<td>96%</td>
<td>95.50%</td>
<td>99.57%</td>
</tr>
<tr>
<td>DoS</td>
<td>96.74%</td>
<td>97.2%</td>
<td>94.31%</td>
<td>99.02%</td>
</tr>
<tr>
<td>Probe</td>
<td>99.07%</td>
<td>85.1%</td>
<td>96.85%</td>
<td>96.71%</td>
</tr>
<tr>
<td>U2R</td>
<td>94.70%</td>
<td>10.1%</td>
<td>84%</td>
<td>56%</td>
</tr>
<tr>
<td>R2L</td>
<td>99.92%</td>
<td>9%</td>
<td>97.69%</td>
<td>97.87%</td>
</tr>
</tbody>
</table>

Table 13: Comparison with models from literature. Measure used: Accuracy. Source: Own edit
Of the compared models, the Bayesian classifier of Chebrolu, Abraham and Thomas (2005) provided the best accuracies for Normal and DoS classes at 0.9957 and 0.9902. The neural network stacking ensemble of this dissertation plan performed best on probe, U2R and R2L classes (0.9907, 0.9470 and 0.9470 respectively), demonstrating that ensemble learning is an approach worth investigating for intrusion detection research.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>0.9781</td>
<td>0.9952</td>
<td>0.9305</td>
<td>0.9943</td>
</tr>
<tr>
<td>DoS</td>
<td>0.9610</td>
<td>0.9826</td>
<td>0.9988</td>
<td>0.9652</td>
</tr>
<tr>
<td>Probe</td>
<td>0.7273</td>
<td>0.7884</td>
<td>0.8677</td>
<td>0.7197</td>
</tr>
<tr>
<td>U2R</td>
<td>0.5714</td>
<td>0.1272</td>
<td>0.1140</td>
<td>0.051</td>
</tr>
<tr>
<td>R2L</td>
<td>0.0875</td>
<td>0.0586</td>
<td>0.6874</td>
<td>0.036</td>
</tr>
</tbody>
</table>

Table 14: Comparison with models from literature. Measure used: Recall. Source: Own edit

Table 14 shows the same analysis structure as Table 13, however the target metric is recall. The available literature that shared recall values per class is even narrower than for accuracy, and many referenced it as detection rate (or used the name detection rate incorrectly for accuracy). Thus only the works of Bouzida et al. (2004), Folino, Pizzuti and Spezzano (2005) and Yao et al. (2017) could be used for recall comparisons.

Bouzida et al. (2004) achieved the best recall for the normal class (0.9952), while Yao et al. (2017) proved to be the best performer for DoS (0.9988), probe (0.8677) and R2L (0.6874). The stacking ensemble only performed well detecting the harder U2R class (at 0.5714 recall). Though when recalls are compared the stacking neural network model performed worse, it managed to provide better predictions for at least one attack type.

5. Conclusion

So far, I managed to collect information to provide adequate answers to two out of three of the research questions I have formulated, also available in section 3.4. The first question reflected on the suitability of machine learning models. They are a suitable approach for detecting intrusions, in fact, according to the literature collected, they are the most effective methods for detecting intrusions. Machine learning models can achieve high performance with detecting DoS and probe attacks and normal activity. U2R and R2L attacks are more complicated. To tackle this challenge, the right course of action is
not the choice of a new model, however. There is no “free lunch” in data science, as there is no single best model which gives perfect predictions. Instead, the scientist needs to strive for ensemble models. My current (and future) comparisons between base classifiers and aggregate classifiers have pointed out the usefulness of this approach.

Speaking of ensembles, the second question put misuse detection, anomaly detection and hybrid detection into perspective. Misuse detection can achieve good results, especially when the models are combined into ensembles, but it does have its limits. In the future, I will focus more on answering this question, particularly, by developing a prototype hybrid intrusion detector. My expectations are to achieve even better classification performance. One idea is to apply encoder networks on normal connections, then evaluate whether it can detect significant difference between normal and attack (or anomalous) traffic. Then use the transformed vectors to train the stacking ensemble consisting of neural networks.

Finally, in my third question I asked about model performance. Simple one-model misuse detection can achieve good results on connections, but the best classifications are generated by composite models, either based on ensemble or hybrid anomaly-misuse detection approaches. From a performance evaluation perspective, the model hyperparameter settings are more decisive than the choice of machine learning model type.

6. Schedule
As of date, I have created two experiments and planned a third one to answer the research questions of section 3.4. Further down the road my plans are the following:

- October 2019: the TensorFlow + KERAS-based update to the stacking neural network model will be implemented, all planned experiments executed. I will present this model on the OGIK 2019 conference in Budapest.

- November 2019 – February 2020: a new intrusion detection model is to be planned implemented and experimented with. The details of this new model are still subject to change. One idea at the writing of this dissertation plan is to utilize autoencoder networks trained strictly on normal connections.

7. Publications


8. References


Author: Csaba Brunner


Expert System Team (2017) *What is Machine Learning - A definition*. Available at:
Author: Csaba Brunner


Author: Csaba Brunner


9. APPENDIX

The conceptual hierarchy used to recategorize attacks and normal behavior in the KDD Cup 1999 dataset. DOS has been set up as part of the normal category in the two-class case because a larger emphasis has been placed on distinguishing the more uncommon attack classes vs DOS and normal. Later, a model can be easily trained to distinguish the DOS class from the normal activity, when necessary.

<table>
<thead>
<tr>
<th>type</th>
<th>five_type</th>
<th>two_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>apache2</td>
<td>DOS</td>
<td>normal</td>
</tr>
<tr>
<td>back</td>
<td></td>
<td></td>
</tr>
<tr>
<td>land</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mailbomb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>neptune</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pod</td>
<td></td>
<td></td>
</tr>
<tr>
<td>processtable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>smurf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>teardrop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>udpstorm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>normal</td>
<td>normal</td>
<td></td>
</tr>
<tr>
<td>ipsweep</td>
<td>probe</td>
<td></td>
</tr>
<tr>
<td>mscan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nmap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>portsweep</td>
<td></td>
<td></td>
</tr>
<tr>
<td>saint</td>
<td></td>
<td></td>
</tr>
<tr>
<td>satan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ftp_write</td>
<td>R2L</td>
<td></td>
</tr>
<tr>
<td>guess_passwd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>httptunnel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>imap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>multihop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>named</td>
<td></td>
<td></td>
</tr>
<tr>
<td>phf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sendmail</td>
<td></td>
<td></td>
</tr>
<tr>
<td>snmpgetattack</td>
<td></td>
<td></td>
</tr>
<tr>
<td>snmpguess</td>
<td></td>
<td></td>
</tr>
<tr>
<td>spy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>warezclient</td>
<td></td>
<td></td>
</tr>
<tr>
<td>warezmaster</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix A: The conceptual hierarchy used to recategorize attacks and normal behavior in the KDD Cup 1999 dataset

Detailed performance measurements for the decision tree bagging classifier. This intrusion detector was implemented using the map-reduce programming paradigm coded using Message Passing interface. Runtime considerations of this approach, however, do not affect classification performance, therefore measurements between parallel setups and sample sizes could be aggregated.

<table>
<thead>
<tr>
<th></th>
<th>4 cores</th>
<th>Small sample (3-5 000 obs.)</th>
<th>Large sample (6-10 000 obs.)</th>
<th>8 cores</th>
<th>Small sample (3-5 000 obs.)</th>
<th>Large sample (6-10 000 obs.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1. run 2. run 3. run 1p2c</td>
<td>1. run 2. run 3. run 1p2c</td>
<td></td>
<td>1. run 2. run 3. run 1p2c</td>
<td>1. run 2. run 3. run 1p2c</td>
</tr>
<tr>
<td>Accuracy</td>
<td></td>
<td>0.785 0.791 0.772 0.796</td>
<td>0.795 0.792 0.809 0.799</td>
<td></td>
<td>0.793 0.782 0.788 0.796</td>
<td>0.797 0.756 0.777 0.799</td>
</tr>
<tr>
<td>Precision</td>
<td></td>
<td>0.959 0.975 0.866 0.967</td>
<td>0.965 0.902 0.903 0.969</td>
<td></td>
<td>0.893 0.931 0.903 0.967</td>
<td>0.936 0.895 0.881 0.969</td>
</tr>
<tr>
<td>Recall</td>
<td></td>
<td>0.483 0.490 0.508 0.506</td>
<td>0.507 0.539 0.585 0.513</td>
<td></td>
<td>0.546 0.493 0.525 0.506</td>
<td>0.529 0.442 0.512 0.513</td>
</tr>
<tr>
<td>F-score</td>
<td></td>
<td>0.642 0.652 0.641 0.664</td>
<td>0.664 0.675 0.710 0.671</td>
<td></td>
<td>0.678 0.644 0.664 0.664</td>
<td>0.676 0.592 0.648 0.671</td>
</tr>
<tr>
<td>AUC</td>
<td></td>
<td>0.735 0.766 0.783 0.793</td>
<td>0.811 0.789 0.776 0.777</td>
<td></td>
<td>0.789 0.719 0.784 0.793</td>
<td>0.772 0.771 0.757 0.777</td>
</tr>
<tr>
<td>4 cores</td>
<td>Small sample (3-5 000 obs.)</td>
<td>Large sample (6-10 000 obs.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------</td>
<td>-----------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1. run</td>
<td>2. run</td>
<td>3. run</td>
<td>1p2c</td>
<td>1. run</td>
<td>2. run</td>
</tr>
<tr>
<td><strong>Accuracy</strong></td>
<td>0.978</td>
<td>0.964</td>
<td>0.981</td>
<td>0.984</td>
<td>0.985</td>
<td>0.985</td>
</tr>
<tr>
<td><strong>Precision</strong></td>
<td>0.477</td>
<td>0.449</td>
<td>0.513</td>
<td>0.511</td>
<td>0.532</td>
<td>0.558</td>
</tr>
<tr>
<td><strong>Recall</strong></td>
<td>0.438</td>
<td>0.438</td>
<td>0.466</td>
<td>0.525</td>
<td>0.469</td>
<td>0.469</td>
</tr>
<tr>
<td><strong>F-score</strong></td>
<td>0.456</td>
<td>0.444</td>
<td>0.489</td>
<td>0.518</td>
<td>0.498</td>
<td>0.510</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8 cores</th>
<th>Small sample (3-5 000 obs.)</th>
<th>Large sample (6-10 000 obs.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. run</td>
<td>2. run</td>
</tr>
<tr>
<td><strong>Accuracy</strong></td>
<td>0.970</td>
<td>0.977</td>
</tr>
<tr>
<td><strong>Precision</strong></td>
<td>0.397</td>
<td>0.467</td>
</tr>
<tr>
<td><strong>Recall</strong></td>
<td>0.421</td>
<td>0.437</td>
</tr>
<tr>
<td><strong>F-score</strong></td>
<td>0.408</td>
<td>0.452</td>
</tr>
</tbody>
</table>

Appendix B: Detailed performance measurements for the decision tree bagging classifier

Detailed performance measurements for intrinsic, time-traffic, host-traffic, content and aggregator models of the neural network stacking ensemble displayed on Appendix C. Each of the base models performed poorly on different classes of the test set. Results were improved by the final or aggregator model.
<table>
<thead>
<tr>
<th>Category</th>
<th>F1 score</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intrinsic</td>
<td>0.842</td>
<td>0.744</td>
<td>0.971</td>
</tr>
<tr>
<td>Normal</td>
<td>0.838</td>
<td>0.998</td>
<td>0.791</td>
</tr>
<tr>
<td>DoS</td>
<td>0.832</td>
<td>0.997</td>
<td>0.731</td>
</tr>
<tr>
<td>Probe</td>
<td>0.826</td>
<td>0.997</td>
<td>0.733</td>
</tr>
<tr>
<td>R2L</td>
<td>0.826</td>
<td>0.997</td>
<td>0.733</td>
</tr>
<tr>
<td>U2R</td>
<td>0.826</td>
<td>0.997</td>
<td>0.733</td>
</tr>
</tbody>
</table>

Appendix C: Detailed performance measurements for intrinsic, time-traffic, host-traffic, content and aggregator models of the neural network stacking ensemble