Improved Anomaly Detection in Computer Networks with Evolutionary Undersampling

Master Thesis

Submitted to the course of studies

Information Technology & Systems Management

at the Salzburg University of Applied Sciences

FH Salzburg

in collaboration with

BGSU®

Bowling Green State University

by

Karl Anton Huber, BSc

Salzburg, September 2020

Head of Degree Programme: FH-Prof. DI Dr. Gerhard Jöchtl

Supervisor (FHS): FH-Prof. Priv.-Doz. DI Mag. Dr. Dominik Engel

Supervisor (BGSU): Ph.D. Robert C. Green II, Ph.D.

Affidavit

I, Karl Anton Huber, BSc, born on 5th June 1989, in Melk, hereby declare that I have written this master thesis entirely on my own without using sources other than the ones explicitly stated. This work is my own and contains no results of collaboration with others, except as specified in the text and my acknowledgements.

Salzburg, September 1st, 2020

Karl Anton HUBER

1810581025

Enrolment number

Acknowledgments

First of all, I would like to thank my supervisor at the University of Applied Sciences Salzburg, FH-Prof. DI Mag. Dr. Dominik Engel, for believing in me and giving me the opportunity to apply for the Marshall Plan Scholarship. I am incredibly grateful for this experience, even if it was overshadowed by the events of 2020.

My thanks also go to Prof. Robert C. Green II, Ph.D. of the Department of Computer Science at Bowling Green State University. As my supervisor at the BGSU, he was always available for a conversation, even if it was mostly virtual. All possible digital communication channels were used during my stay. His guidance, exciting discussions, and constructive feedback were incredibly important for the success of this work. Fortunately, we had the opportunity to meet at least a few times before the university was completely closed down.

A sincere thank you to my roommates Chase Tipsword, Jeremy Strader, and Matthew Horton. Thankfully, they always found a way to keep me sane during the quarantine period, which hit the United States harder than any other country on this planet.

Finally, I would like to express my appreciation to my family and especially to my partner Jennifer Brunner for their support and encouragement during the five years of my studies, especially in times of doubt.

Thank you.

General Information

Author:	Karl Anton Huber, BSc
University:	Salzburg University of Applied Sciences
Degree Program:	Information Technology & Systems Management
Title of the Thesis:	Improved Anomaly Detection in Computer Networks with Evolutionary Undersampling
Keywords:	Anomaly Detection Pattern Recognition Imbalanced Data Evolutionary Sampling Over- and Undersampling Statistical Testing Python
Supervisor at FHS:	FH-Prof. Priv.-Doz. DI Mag. Dr. Dominik Engel

Abstract

This work investigates if the evolutionary undersampling method is able to compete with other state-of-the-art undersampling methods for real-world, imbalanced datasets. Anomaly-based intrusion detection systems depend heavily on sampling as malicious network traffic appears primarily in an unbalanced way, thus, causing low detection rates when using standard anomaly detection approaches. As an example of real-world intrusion detection, the Australian Defence Force Academy Linux Dataset is used, because it includes contemporary attacks and is well-known for evaluating the performance of anomaly-based intrusion detection systems. Evolutionary undersampling (EUS) is thoroughly explained herein, and then the implementation is used to evaluate its performance using the ADFA-LD dataset. Finally, the performance is compared to other well-known undersampling methods by using appropriate statistical tests. EUS shows promising results, and with most performance measures, there is no statistically significant difference between it and traditional undersampling methods.

Table of Contents

List of Abbreviations

ACC	Accuracy
ADASYN	Adaptive Synthetic
ADFA-LD	Australian Defence Force Academy - Linux Dataset
AIDS	Anomaly-based Intrusion Detection System
ALLKNN	AllK-Nearest-Neighbor
ANN	Artificial Neural Network
AUC	Area Under the ROC Curve
C-NN	Condensed-NearestNeighbor
CNN	Convolutional Neural Network
DT	Decision Tree
EA	Evolutionary Algorithm
EBUS	Evolutionary Balancing Undersampling
ENN	EditedNearestNeighbor
EUS	Evolutionary Undersampling
EUSCM	Evolutionary Undersampling Guided By Classification Measures
FDR	False Discorvery Rate
FN	False Negative
FNR	False Negative Rate
FP	False Positive
FPR	False Positive Rate
FWER	Family-wise Error Rate
GA	Genetic Algorithm
GCP	Google Cloud Platform
GM	Geometric Mean
GS	Global Selection
HIDS	Host-based Intrusion Detection System
IDE	Integrated Development Environment
IDS	Intrusion Detection System
IT	Information Technology

KNN	K-Nearest-Neighbor
LDA	Linear Discriminant Analysis
MDC	Minimum Distance Classifier
MLP	Multi Layer Perceptrons
MS	Majority Selection
NCR	NeighborhoodCleaningRule
NIDS	Network-based Intrusion Detection System
OSS	OneSidedSelection
QDA	Quadratic Discriminant Analysis
RENN	RepeatedEditedNearestNeighbor
ROC	Receiver Operating Characteristics
RUS	RandomUnderampler
SIDS	Signature-based Intrusion Detection System
SMOTE	Synthetic Minority Oversampling TEchnique
SVM	Support Vector Machine
TL	TomekLinks
TN	True Negative
TNR	True Negative Rate
TP	True Positive
TPR	True Positive Rate

List of Figures

List of Tables

1. Introduction

Detecting malicious activity in computer networks reliably is a topic that has received increasing attention in recent years. With ever-increasing connectivity, the need to detect attacks has become more critical, especially as more parts of critical infrastructure, such as energy networks, production facilities, or automotive infrastructure, are included in the move towards digitalization and thereby connected to the outside world. Therefore, in addition to traditional countermeasures such as firewalls and network segmentation, it is of utmost importance to detect attacks, ideally in real-time. To achieve this, there are two sectors in network intrusion detection that can be categorized as *anomaly detection* and *signature detection*. The difficulty in anomaly detection is categorizing the traffic into "normal" and "abnormal" behavior. Typically, malicious behavior and network intrusions represent a minimal subset of all network traffic, which leads to a considerable imbalance in classification. As argued by the authors of [1], this imbalance increases the difficulty of classifying the data correctly. Misclassified "normal" network traffic (or false positives) is a considerable problem for imbalanced datasets, since those false positives lead to data fragmentation, packet loss of relevant packages, and noise. Furthermore, a high number of false positives may lead to a non-negligible number of false alarms for the operator. With the digitalization of critical infrastructures, it is essential to filter alerts to ensure the operator's full focus. An intrusion detection system that produces a high number of false positives drains the focus of the operator and, therefore by itself poses a security risk due to the high number of alarms. Hence, it is imperative to significantly decrease intrusion detection systems' false positive rate, especially in the context of digitalization.

However, making this detection reliable is a challenging task because, in relation to non-malicious traffic, the amount of traffic that is malicious is a minute fraction. Simply put, this is a classification problem, where the number of samples of one class is significantly higher than the number of samples of the other class. Therefore, almost all anomaly-based intrusion detection approaches currently used suffer from a high false-positive rate. In order to improve the accuracy and detection rate of malicious network traffic, different machine learning methods are applied, focusing on addressing the predominant imbalance problem. Therefore, the classification approach needs to distinguish classes of very different cardinalities: the *majority* or *negative* class, and the *minority* or *positive* class. When discussing intrusion detection systems, the majority class is referred to as the normal or benign network traffic with no malicious intent. In contrast, the minority class stands for malicious traffic that needs to be identified as fast as possible. This identification is usually achieved using a classifier whose main objective is to minimize the false-positive and false-negative rates.

There are two techniques to reduce the imbalanced data distribution in these datasets: over- and undersampling. Oversampling tries to replicate or generate new examples from the minority class while undersampling reduces data by eliminating majority class examples. Both pursue the same goal of equalizing the number of examples in each class. Various sampling methods have been shown to be successful and are widely used; however, every one of these methods also includes drawbacks, so the goal is to improve the performance of traditional methods for imbalanced datasets. One of the most promising techniques to overcome these drawbacks is evolutionary undersampling [2].

From these findings the main research question of this thesis can be formed: **Is it possible to implement an evolutionary undersampling algorithm with state-of-the-art genetic algorithms that is able to compete or improve the performance of commonly-used undersampling methods in the field of intrusion detection?**

This thesis will first start with a thorough investigation of state-of-the-art intrusion detection and pattern recognition methods. Then, the class imbalance problem will be presented and explained why it is relevant in this problem domain. Solutions to this imbalance problem will be described, and subsequently, the performance evaluation will be shown. The next chapter will deal with the methodology used to implement the proposed evolutionary undersampling (EUS) algorithm. The idea is to use EUS as a first stage in intrusion detection to decrease the degree of imbalance, which should result in an improvement of the performance of anomaly-based intrusion detection methods. Within this chapter, the general idea behind evolutionary algorithms and background of EUS will be explained. Then, statistical tests necessary to check any significant differences between undersampling algorithms are investigated. The used dataset will conclude the theoretical part of this thesis. The implementation is then introduced by describing the environment used for development. Using the theoretical foundations, an implementation of EUS will be presented, and its performance calculated with selected classifiers. Next, the results are compared with different, well-known undersampling methods. Finally, these results are used for statistical testing to determining if EUS has any significant differences compared to those other sampling methods.

2. Literature Review

In this part of the thesis, underlying principles of anomaly detection are explained. Within this first part, the interdependence of the following chapters can be shown in Figure 2.1. Some parts of cybersecurity would not be possible without an intrusion detection system (IDS), whereas an IDS would not be able to make the right decision without its underlying pattern recognition system. The decisions made by pattern recognition systems depend largely on the classifier and sampling method used. By providing the interconnected and well-adjusted parts, an IDS is able to deliver satisfying performance without raising too many false alarms or missing out on potential threats.

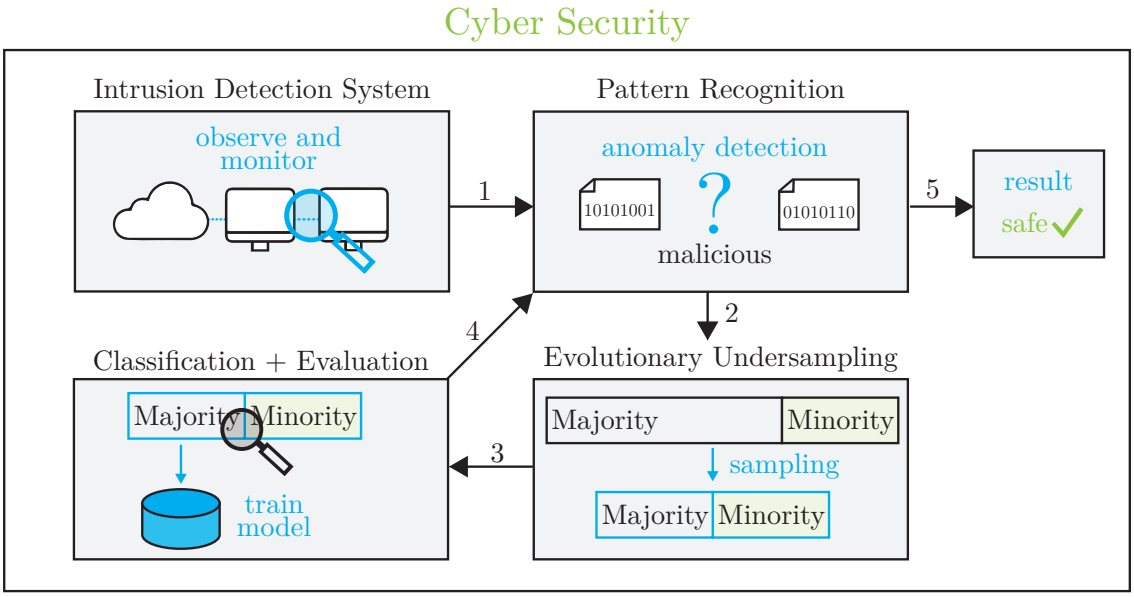


Figure 2.1.: Connection between the following chapters.

This chapter provides insight into the definition of information security in Section 2.1, and presents the idea behind IDS in Section 2.2. In Section 2.3, state-of-the-art pattern recognition background will be explained, and the approaches and methods, the steps in pattern recognition, and classification are shown.

2.1. Information Security

Bruce Schneier, who is often referred to as the closest thing to a rock star in the field of cybersecurity, mentioned: *"The internet can be regarded as the most complex machine mankind ever built. We barely understand how it works, let alone how to secure it"* [3]. This statement was phrased over ten years ago, and with the ever-increasing speed of technological development, it has never been more true than it is today.

In a general sense, security means the protection of enterprises, individuals, and information from intentional attacks, breaches, incidents, and consequences. It is desired to protect these assets against the most likely forms of attacks to the best and reasonable extent. In this context, reasonable means that assets should only be protected to the extent that these assets' productivity remains at an acceptable level. The best protection of an asset is only worthwhile so long as the asset remains productive after all protective mechanisms have been put into place. Additionally, when securing an asset, it also must be considered on how that level of security relates to the value of the protected item. The cost of security put into place should not exceed the value of the protected asset. Ultimately, the goal of security is to find the balance between protection, usability, and cost [4]. The authors of [5] identified several terms that describe this protection of assets, including computer security, network security, information security, and cybersecurity. The latter two terms are the most commonly used in the literature, and are often used interchangeably. As illustrated in Figure 2.2, cybersecurity can be seen as a component of information security. More specifically, information security deals with information, regardles of what form it takes, whether it is a paper document, intellectual property, or verbal and visual communication. Cybersecurity, on the other hand, deals with the protection of digital assets, including everything from hardware to networks, and the information that is processed, stored, and transferred over these Internet-linked systems.

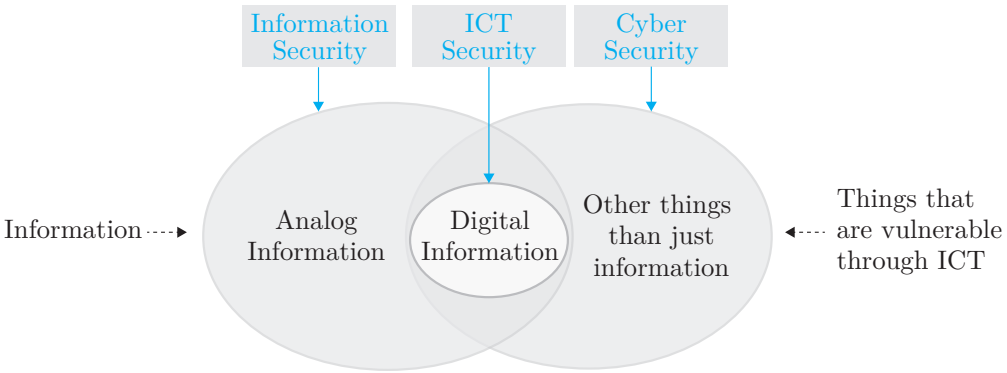


Figure 2.2.: Relationship between ICT-, cyber- and information security, by [6].

Due to the increasingly complex nature of information in current times and the rapid development of technologies, the term "cyber" is often used too widely by marketing, vendors and analysts. The increasingly complex and networked nature of information and critical infrastructures in the digital age is challenging to understand fully. Using new technologies has introduced a host of new vulnerabilities with far-reaching implications. As technology moves forward with time, security demands change, and introduces the shift

from information security to cybersecurity. In [5, p.5], cybersecurity is defined as "the protection of information assets by addressing threats to information processed, stored and transported by inter-networked information systems."

Nowadays, the objective of information security consists of the three critical components of the Confidentiality-Integrity-Availability (C-I-A) triangle, as illustrated in Figure 2.3. Confidentiality aims to protect information from unauthorized access or disclosure. The confidentiality levels vary for different types of information and may change over time, for example, financial, medical, and personal data may require a higher degree of confidentiality than other information. Integrity, on the other hand, is the protection of information from unauthorized modification during the exchange or transmission. The last component, availability, deals with timely and reliable access to the required information on request.

Figure 2.3.: Cybersecurity triad, by [5].

Using the elements of the CIA triad, security issues can be discussed in a very specific fashion. To clarify the general understanding of these elements, the example of a lost delivery is used in [4]. Suppose a security problem occurs when the only existing backup of unencrypted, sensitive data tapes is lost. As for confidentiality, there is a problem because those tapes were not encrypted to begin with. From an integrity standpoint, there is an issue because, after the possible recovery of these tapes, no one would immediately know if the files were altered or not. Availability wise, the issue would arise unless the tapes were recovered because it was the only copy of the files.

In order to mitigate risks in information technology (IT) systems, there are various methods to ensure that a given type of threat is accounted for. These methods are often referred to as "controls", and can be divided into three categories: physical, logical, and administrative. Physical controls protect the physical environment in which the asset is located, including fences, gates, locks, and cameras. Logical controls protect the logical systems, networks, and environments that process, transmit, and store data. These assets include items such as passwords, encryption, logical access controls, and firewalls. The authors in [7] referred to these items as the firstline of cybersecurity measures, as they are the first real entry point into the cybersecurity domain. Finally, the administrative controls are based on rules, laws, policies that define how users should behave in the protected environment [4].

Since this thesis deals with intrusion detection, only the logical control is dealt with in more detail. It is an integral element to monitor data and information flowing into and out of an organization. The most commonly used instrument for detecting threats are logs from various sources such as IDS, anti-virus software, firewalls, and proxies. Those devices are implemented at the borders of, and within, computer networks to increase security. Therefore, monitoring is considered carefully within this thesis. The next section explains the concept of an IDS in more detail.

2.2. Intrusion Detection Systems

IDS are considered to be the second line of security measures, as they work in conjunction with various network devices by monitoring network usage anomalies. These systems operate in the background continuously and notify administrators when an intrusion is detected. An intrusion is defined as any kind of unauthorized activity that can damage the protected system [8]; therefore any attack that poses a possible threat to the CIA triad is considered to be an intrusion. For example, events that would render computer services unresponsive to authorized users are considered an intrusion. As IDS is usually situated right after firewalls, it is the main goal of an IDS to identify malicious network traffic and computer usage that passed the firewall undetected. It is vital to achieving high protection against actions that compromise the CIA triad of computer systems. Depending on how an IDS identifies those intrusions, they can be categorized into two groups: Signature-based Intrusion Detection System (SIDS) and Anomaly-based Intrusion Detection System (AIDS).

SIDS, commonly called misuse-based IDS, are based on techniques that match a pattern to an existing known attack. When an intrusion event signature matches the signature of an intrusion that is already stored in a signature database, an alarm event is triggered. These pattern matching techniques result in excellent detection accuracy for previously known attacks. In the literature [8], they are also labeled as “Knowledge-Based Detection” or “Misuse Detection”. A detection rule could resemble a simple if-else condition, for example, “if (source IP address=destination IP address) then label as an attack.” Unfortunately, many intrusions are more complex than a simple IP address comparison. As modern attacks and malware become more sophisticated, storing and extracting information over multiple packets is often necessary, resulting in more resource requirements. Signature-based methods have difficulties detecting zero-day attacks because no matching signature is stored in the matching database until the signature of a new attack is identified, extracted, and stored. Even though targeted zero-day attacks are declining [9], they are still present, with almost four billion tracked cases in 2017, rendering SIDS techniques progressively less effective.

Contrary to the previous technique, AIDS depend on a normal model of the behavior to decide whether an event is malicious. This model is created by using machine learning, statistical-based, or knowledge-based pattern recognition methods, which will be explained thoroughly in Section 2.3. If there is a significant deviation between the observed behavior and the model created, the behavior is assumed to be an anomaly. Anomaly-

based systems assume that any abnormal behavior that differs from standard behaviors is classified as an intrusion. AIDS are designed in two phases: the training and the testing phase. During the training phase, all network traffic in an allegedly clean environment is used to create a normal behavior model. The testing phase is where current traffic is compared with the model created in the training phase. By scanning for abnormal behaviors and not relying on a signature database, AIDS can detect zero-day attacks; the main advantage it has compared to a signature-based system. Furthermore, AIDS can also detect internal malicious activities; if an intruder starts executing allowed events in an unusual way, the systems will most likely raise an alarm [8].

Usually a combination of signature and anomaly-based IDS provides the best detection rates. To get an better overview of available IDS types, Table 2.1 shows the differences between SIDS and AIDS.

	Misuse Detection	**Anomaly Detection**
Detection performance	Low false alarm rate; High missed alarm rate	Low missed alarm rate; High false alarm rate
Detection efficiency	High, decrease with scale of signature database	Dependent on model complexity
Dependence on domain knowledge	Almost all detections depend on domain knowledge	Low, only the feature design depends on domain knowledge
Interpretation	Design based on domain knowledge, strong interpretative ability	Outputs only detection results, weak interpretative ability
Unknown attack detection	Only detects known attacks	Detects known and unknown attacks

Table 2.1.: Differences between misuse detection and anomaly detection by [10]

Another way of categorizing IDS can be made with respect to the input data sources used to detect abnormal activities. There are two types of technologies that can be considered in terms of data sources: Host-based Intrusion Detection System (HIDS) and Network-based Intrusion Detection System (NIDS). Figure 2.4 illustrates the different positions of both methods.

When network traffic is extracted from a network through packet capture, NetFlow, or other network data sources, the system is called a NIDS. These systems can monitor many computers and devices within a network, but it is crucial to place the NIDS in a location where it is not being overloaded. This problem arises by the high bandwidth of modern high-speed communication networks, as all traffic is usually redirected to the network intrusion detection system. By placing the NIDS behind filtering devices such as firewalls, this problem can be alleviated and decrease the traffic that the system needs to inspect [8]. While dealing with large amounts of traffic, they generally only perform a relatively superficial inspection to decide whether the packets are malicious or normal.

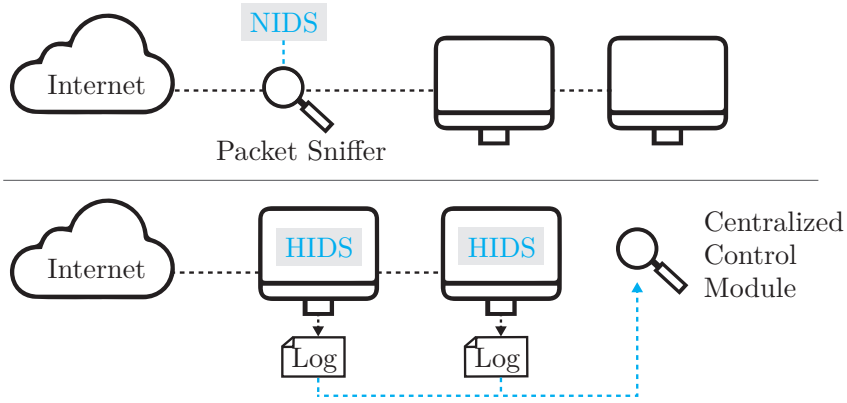


Figure 2.4.: Network-based IDS vs. Host-based IDS

Further, there are HIDS. These systems are located directly on the host systems and inspect sources including event logs, firewall logs, application logs, and database logs. Host-based systems can detect intrusions that do not involve network traffic or intrusions that are directed at the network interface of a particular host. Their scope is considerably reduced compared to network-based systems, as they only monitor one single host at a time. They can detect changes in executable applications, detect the deletion of critical files, and issue warnings when privileged commands are carried out [4]. The advantages and disadvantages of such systems can be found in Table 2.2.

	Host-Based IDS	**Network-Based IDS**
Source of data	Logs of operating system or application programs	Network traffic
Deployment	Every host; Dependent on operating systems; Difficult to deploy	Key network nodes; Easy to deploy
Detection efficiency	Low, must process numerous logs	High, can detect attacks in real time
Intrusion traceability	Trace the process of intrusion according to system call paths	Trace position and time of intrusion according to IP addresses and timestamps
Limitation	Cannot analyze network behaviors	Monitor only the traffic passing through a specific network segment

Table 2.2.: Differences between host-based and network-based IDS by [10]

The authors of [7] describe the actions of an IDS in the following way: IDS are capable of analyzing HTTP packets, IP flow records, DNS replies, and Honeypot data. HTTP

traffic, which represents a significant portion of Internet users' traffic, can be used by analyzing the uniform resource identifiers (URLs) embedded in HTTP packets to help prevent malicious communications. For instance, IP flow records can be used to provide invaluable data for highlighting botnet communications or detecting intrusions. These records can help to trace every communication from the Internet to enterprise networks and vice versa. Monitoring the Domain Name System (DNS) requests, an IDS is able to efficiently and proactively identifying malicious domains. A honeypot is commonly used to emulate a vulnerable service in a production network. Behind this fake vulnerability, there is often some fake production data hidden so that the attacker thinks he penetrated the security measures. Logging honeypot information helps to obtain information about attackers and their behaviors. These behaviors could contain specific network protocols that are targeted the most, IP addresses used, and scanning strategies.

Figure 2.5 presents a taxonomy of IDS proposed in [10]. It shows two types of IDS classification methods, including the data source as the main classification consideration, and the detection method as a secondary classification element.

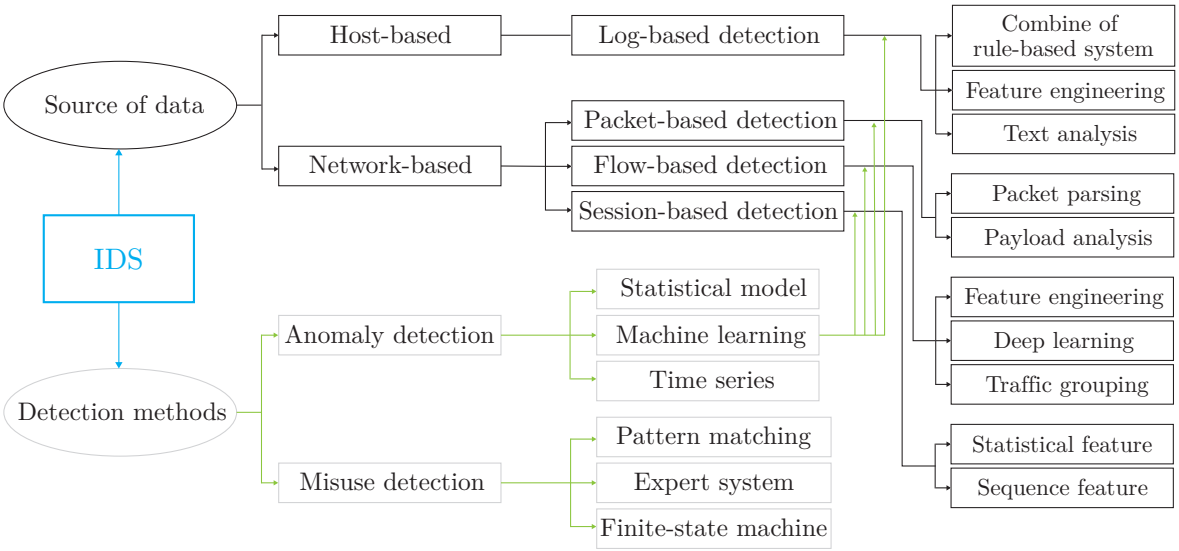


Figure 2.5.: IDS general overview by [10]

This thesis focuses on anomaly-based intrusion detection systems. To explain the underlying principles, the next section covers pattern recognition methods necessary during this detection process.

2.3. Pattern Recognition

Usually, pattern recognition is a problem of differentiation between classes. To explain the general steps of pattern recognition in an understandable way, the authors of [11] use this example: Thousands of people within a population should be divided into four types: tall and thin, tall and thick, small and thick. Therefore, each person needs to be *classified* in

one of the four populations. This recognition process turns into *classification*. To decide which class the person belongs to, *features* are needed to make the right decision. In order to decide if a person is tall or small, age would be the wrong feature to choose; instead, the *feature selection* process could select a combination of height and weight. The act of obtaining these measurements is called *feature extraction*. If the feature extraction process is too difficult to perform, alternative features need to be found, or the existing data needs to be transformed into usable information. This process is called *preprocessing*.

Although pattern recognition is a natural and straightforward process for humans to recognize voices or faces, these tasks can be very demanding and complicated for machines [12]. When trying to translate the above example to the cybersecurity domain, a pattern to be recognized could be a malicious attack. In order to find this pattern, all the steps of the pattern recognition need be passed through: first, the general approaches and methods, and the necessary steps will be explained in the next two subsections. Then, the next section is dedicated to available models and their classification tasks. Finally, this chapter will end with an evaluation of the classification approaches.

2.3.1. Approaches and Learning

In pattern recognition, machine learning, and datamining, one major task is to construct good models from datasets. Generally, a *dataset* consists of *feature vectors*, where each feature vector is a description of an object by using a set of features. *Dimensionality* describes the number of features in a dataset. If a feature vector is formed of two features, its *dimensionalty* is two. Ultimately, a *model* is a predictive model or a model of the structure of the data that the pattern recognition process wants to construct or discover from the dataset. These structure models can include decision trees, neural networks, support vector machines, etc. The generation process of these models is called learning or training. [13]. There are four basic approaches to create models in pattern recognition: template matching, statistical approach, syntactic or structural approach, and neural networks [14].

Template matching is one of the simplest and earliest to pattern recognition. It matches the pattern to be recognized with an existing database of patterns, and can be compared with signature-based IDS approaches, where the pattern is already known and stored in a database. While useful in some application domains, it has several disadvantages. For instance, it fails to find patterns that are distorted or altered in any way. Also, if the pattern to be recognized is not already stored in the database, this approach will not be able to recognize the corresponding class.

Statistical approaches focus on feature vectors. Features are specified by the investigator and play an essential role in the effectiveness of the approach. Depending on how efficiently the features were selected, the classification process may change drastically. Thus, it is essential to find and extract the correct features to distinguish between different classes. The primary goal of this is to find well-formulated decision rules within the feature space to maximize the ability to distinguish between the different classes in a dataset.

The syntactic or structural approach is used when the recognition problem involves more complex patterns. This approach uses a hierarchical perspective where patterns are viewed as a composition of simple subpatterns. These subpatterns are composed of even simpler subpatterns. The simplest subpatterns that can be recognized are called "primitives". In syntactic pattern recognition, there is a formal analogy drawn between a language's syntax and the structure of patterns. The primitives are the alphabet of the language, and the patterns resemble sentences belonging to a language. When used correctly, an extensive collection of intricate patterns can be described by using grammatical rules and a small number of primitives. The grammatical rules can be inferred from available training data. The syntactic approach can lead to a broad diversity of possibilities, but with the cost of great computational effort and by requiring large training sets [14].

Neural networks have the ability to learn more complex nonlinear relationships between their input and output. The way biological nervous systems, such as the brain, process information, are the main inspiration of neural networks. Using large amounts of interconnected processing units, neural networks can efficiently perform difficult tasks. Other characteristics include that they use sequential training procedures and adapt themselves to the provided data. This last property is one of the main reasons why neural networks are becoming increasingly popular for pattern recognition problems; they have a seemingly low dependence on domain-specific knowledge. Furthermore, there is a high availability of efficient learning algorithms for users.

Learning Techniques

The process of generating models from data is called learning or training. There are different learning settings in pattern recognition, among which the most common are *supervised learning*, *unsupervised learning* and *reinforced learning*. In *supervised learning*, a pattern recognition method learns based on a labeled dataset, meaning that the dataset provides the correct answer to the classifier in the form of a targeted label [13]. It is a form of learning, a task under supervision; someone is present judging whether the answer is right or wrong. If the label is categorical (a shape, or positive/negative), the task is called *classification*, and the learner is called *classifier*. However, if the label is numerical (a range of infinite numbers e.g., length including decimals) the task is called *regression* and the learner is called *fitted regression model*. For instance, a labeled dataset of flower images would tell the model which photos were lilies, roses, and tulips. When shown a new image, the model would compare it to the training examples to predict the correct label [15].

Unsupervised learning, on the other hand, is when the dataset has no assigned targets. The pattern recognition method tries to make sense of the data by extracting features and patterns on its own. The results of this technique are called *clustering* (discovering groups) or *density estimation* (determine the distribution). It is often challenging to obtain datasets that are perfectly labeled, making unsupervised learning methods key in various research areas. However, they come with the disadvantage that it is difficult to measure the accuracy of these learning methods because there is no ground truth in unlabeled datasets.

The third technique, *reinforced learning*, is an iterative process of finding the best solution for a particular goal or improving the performance on a specific task. In order to make choices, this learning method tries different variations of a specific task. It improves the performance with the help of experiences from earlier tries and exploration of new tactics [15].

As this thesis deals with intrusion detection, using a dataset where all targets assigned to the correct classes, the supervised learning method is considered for the following parts. Subsequently, these approaches and methods need to go through various steps to work as expected. This will be the focus of the next Section 2.3.2.

2.3.2. Steps in Pattern Recognition

After introducing the basic pattern recognition approaches and learning techniques, this section will outline the necessary steps to construct a good model. The central focus of this thesis lies on statistical pattern recognition approaches and, as mentioned before, supervised learning approaches. Figure 2.6 shows a simplified pattern recognition procedure to receive an optimal response for a given input.

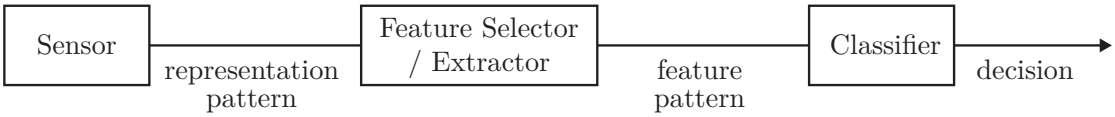


Figure 2.6.: Pattern classifier by [12]

This response is an estimate of the class to which the design belongs. In intrusion detection, these classes are usually divided by normal or malicious behavior (negative or positive class). First, a sensor collects data for a given problem. Then the data may undergo several transformation stages before the classifier provides its decision. These transformations stages, including preprocessing, feature selection, or feature extraction, process the data in different ways and have a number of functions. They can reduce the number of features in a feature vector, which is called *dimension reduction* or *feature selection*, and can also remove redundant or irrelevant information or transform the vector into a more appropriate form for subsequent classification. It is a goal to find the intrinsic dimensionality; i.e., to find the minimum number of variables (features) required to capture the structure within the data. In many cases, it is necessary to perform at least one or more transformations of the measured data to get a good estimation. After these transformation stages are complete, the classifier can investigate the data. There are various classifiers that can be constructed from a dataset, such as linear discriminant functions, decision trees, and support vector machines. These different classifiers will be examined in Section 2.3.3.

As mentioned above, Figure 2.6 is simplified, and the pattern recognition process consists of more phases than those shown in the figure. The enumeration below shows a fairly typical process, even though not all the stages may be present or some may be merged [12]:

1. **Problem formulation**: Trying to fully and clearly understand the given problem that has to be solved and planning the next steps.
2. **Data collection**: Collecting or creating measurements on appropriate variables. Storing additional details of the data collection procedure, such as the ground truth (label data for supervised learning).
3. **Initial examination**: The first look at the collected data, plotting data, checking random samples, and calculating statistics in order to get a better understanding of the structure.
4. **Feature selection** or **feature extraction**: Picking needed measurements that are appropriate for the pattern recognition task. Feature extraction might generate variables or measurements by linear or nonlinear transformations of the gathered dataset. Sometimes classification and feature extraction may be summarized in one step, as some classifiers include the optimization process of the extraction phase into their design.
5. **Unsupervised pattern classification** or **clustering**: This is explorative data analysis and may already allow for successful completion of a study. It may also be used as another means of preprocessing the data for a supervised classification procedure.
6. **Use discrimination or regression methods**: Methods are used as appropriate. This step provides a training set of exemplar patterns and is used to train the classifier.
7. **Evaluation of the results**: Applying the trained classifiers to independently test data with labeled targets. The performance of classifiers is often summarized by using a confusion matrix (More in Section 3.3).
8. **Interpretation**

These steps can be an iterative process, as the analysis of the results may produce new hypotheses that require more data collection than initially assumed. This cycle may be aborted during the first iteration, or it may have to be aborted because it is determined that the data may not be able to answer the original question. If the question cannot be answered, the problem must be reformulated. To show the bigger picture of pattern recognition and all of its steps, Figure 2.7 is used to illustrate this iterative process. The blue highlighted steps show the used classification categories this thesis will present.

After explaining the steps and approaches in pattern recognition, the next chapter will give a brief overview of classifiers and models, and how decisions are made.

2.3.3. Classification

To explain the fundamentals of classification, an example from [16] is used. The authors of this example try to let a classifier distinguish between two different types of fish. In this example, the feature vector is two-dimensional because it consists of two different

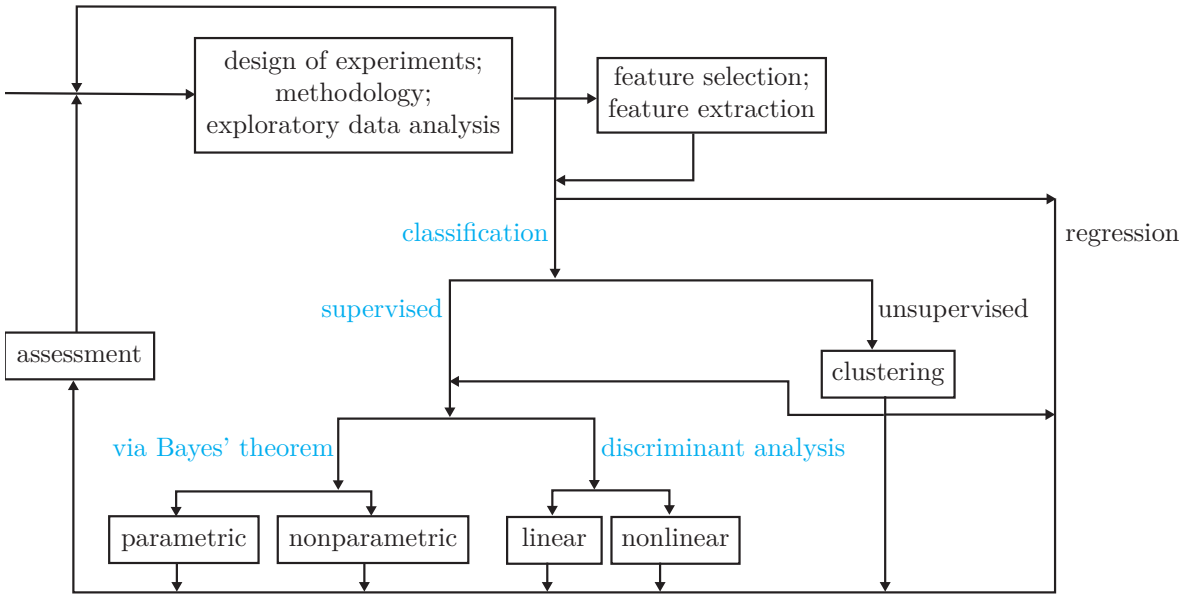


Figure 2.7.: The pattern recognition cycle by [12]

features (lightness and width). The feature extraction process defined that these two features are sufficient to differentiate between both species. It is challenging to minimize the needed features of an object to be still able to classify a class as reliable as possible. The classifier's main goal is to mathematically describe a decision boundary between the two classes that can decide whether the sample is class A or class B. The plot illustrated in Figure 2.8 shows the calculated decision boundary, represented by the black line, in the unseen test data. The rule suggests classifying the fish as salmon if the feature vector falls beyond the decision boundary and as a sea bass otherwise. Besides a handful of fish on the wrong side of the classification boundary, this rule appears to do an excellent job of separating the samples.

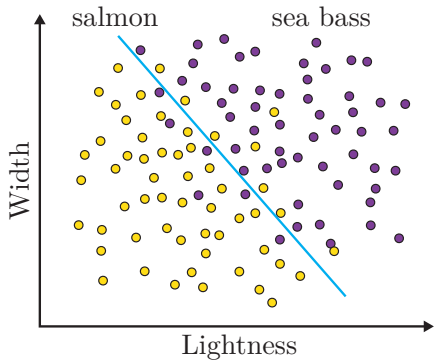


Figure 2.8.: Classifier decision rules by [16]

In order to improve the decision boundary, even more features could be added; if the computational cost in attaining more features would not matter, the decision boundary could be more complicated than the simple straight line. Figure 2.9 illustrates that a more

complex decision rule would lead to perfect classification with the trained data samples; however it is of note that this method would have considerable problems when dealing with novel datasets. In this example, new samples, such as the red dot in the middle of the dataset, would most likely be misclassified as a sea bass even though it would more probably be a salmon. This behavior is known as *overfitting*.

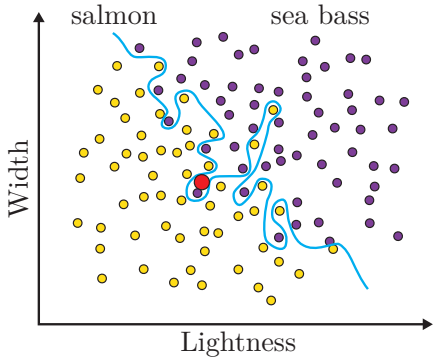


Figure 2.9.: Overly complex classifier decision rules by [16]

Classifiers should also work on known and novel patterns; the above mentioned "solution" would be not desirable, and the satisfaction of the excellent performance would be premature. This is an issue of *generalization*. It is very unlikely that the complex decision boundary would provide good generalization when presented with a new dataset as it is "tuned" to the particular dataset of salmon and sea bass. One approach to overcome this overfitting issue would be to simplify the decision boundary to get a slightly poorer performance on the training samples, but therefore a better performance on new datasets [16]. To find the perfect trade-off between a classifier's simplicity and the performance on unseen data, the test data could be preprocessed in a special way; this is called *data splitting*.

Data splitting usually divides the dataset into two parts: training data and test data (Figure 2.10, step 2). In supervised learning methods, the training dataset is used to train the classifier to generate the decision boundary. The trained classifier can then be compared with the correct target labels for each input vector in the training dataset. Based on the results of the comparison and the specific classifier used, the parameters can then be adjusted (Figure 2.10, step 3). Successively, the fitted classifier or model is used to predict the classes in the test data (Figure 2.10, step 4). This dataset represents the unseen part of the evaluation and is unbiased to the previous training data [13, 16]. Further explanation of different data splitting methods will follow in Section 3.3 of this thesis.

As mentioned above, the main goal of statistical pattern recognition approaches is to find decision rules within the feature space to maximize the ability to distinguish between the different classes. In order to create well-suited decision boundaries, the two main approaches are shown in Figure 2.7: Bayes' theorem and discriminant analysis. In reality, there are three approaches; ensemble methods combine classifiers of these two main approaches to utilize the advantages of both while expecting better generalization results.

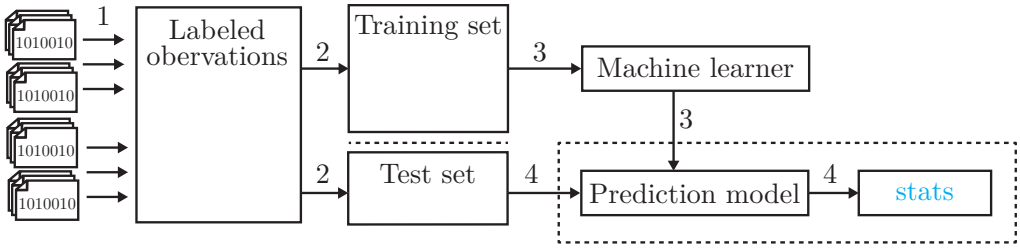


Figure 2.10.: Supervised learning steps from labeled data from [15]

The next part will briefly describe the basic differences between these methods.

Elementary Decision Theory

This approach focuses on discrimination that is based on knowledge of each class's probability density function in the dataset. To accomplish this, the Bayes theorem is used to find the probability of a class based on prior knowledge of conditions that might be relevant to the event. There are two terms used in this theorem that can help with understanding this topic: *a priori* probability and *a posteriori* probability. Priori is the probability of an event happening before any additional knowledge is earned. Posteriori probability, on the other hand, is the probability obtained after we have observed a decision [17]. For instance, the optimal Bayes' decision rule tries to assign a pattern to the class with the highest posterior probability. Since it is not possible to calculate the posteriori probability for each class, as the necessary knowledge is not always available, an estimated density function can be built. Classification is achieved by applying the Bayesian decision rule. For this, knowledge from the class-conditional density function $p(x|w_i)$ (normal distributions that are estimated from the data), or non-parametric density estimations (kernel density estimation) is required [12]. Commonly used classifiers utilizing the Bayes' theorem are the k-Nearest-Neighbor (kNN) classifier and the Parzen classifier.

Discriminant Analysis

Compared to the latter approach, the discriminant analysis uses discriminant functions instead of probabilities to determine its decision rules. A discriminant function is a function of the pattern x that leads to the desired classification rule. The authors of [12] describe that in a two-class problem, a discriminant function $h(x)$ with a threshold k is denoted as:

$$\begin{aligned} h(x) > k \Rightarrow x \in \omega_1 \\ h(x) < k \Rightarrow x \in \omega_2 \end{aligned} \tag{2.1}$$

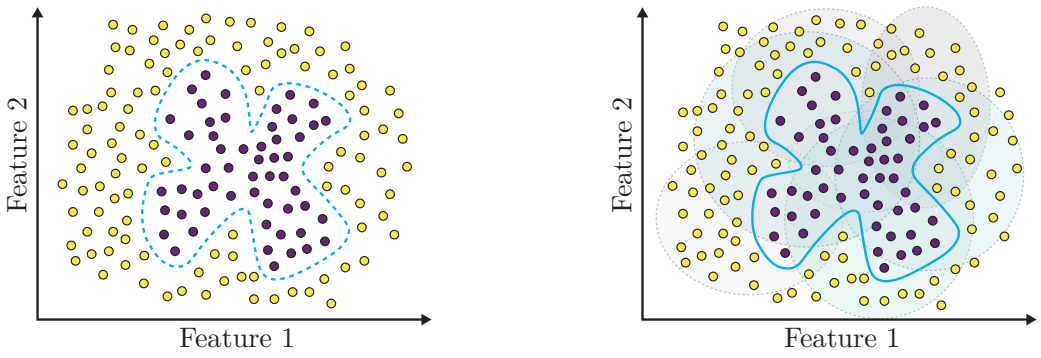
If the case occurs that $h(x) = k$, then the pattern x is assigned arbitrarily to one of both classes. The most crucial difference between Bayes' theorem and discriminant analysis is that the form of the discriminant function is differentially influenced by both. In the Bayes' theorem, the form of the discriminant function is specified and imposed by the

underlying distribution; in the discriminant analysis, the distribution only specifies the function and not the form.

Three discriminant approaches can be distinguished: linear functions, non-linear functions (kernel-based or projection-based approaches), and tree-based approaches. Conventional classifiers using discriminant analysis are Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), Minimum Distance Classifier (MDC), and Support Vector Machine (SVM) [12].

Ensemble Methods

Ensemble methods use multiple learning algorithms to solve one classification problem. They combine the results from different ordinary learning approaches to a set of learners. This is depicted in Figure 2.11. An ensemble is constructed by combining several learners, called base learners. These base learners have to be as accurate as possible and as diverse as possible [13, 18], therefore, the strategy is to create many classifiers and combine their outputs in a way that the combination of the results improves the overall performance. However, this requires the individual classifiers to make different errors on different occasions in the dataset. The idea here is that the strategic combination of those different errors of each classifier reduces the total error. This concept is commonly used with low pass filtering of the noise. If each classifier is as unique as possible, especially with respect to misclassification, this set of classifiers is said to be *diverse* [18].



(a) (b)

Figure 2.11.: Examples of an ensemble of individual classifiers forming a complex decision boundary, from [18]

Figure 2.11a shows a decision boundary in a two dimensional, two-class problem with a complex decision boundary. A traditional linear classifier would not be capable of learning this complex boundary; however, a combination of circular classifiers, the ensemble, would classify smaller sub-parts of the dataset. In a sense, it follows a divide-and-conquer approach by separating the feature space into smaller, easier partitions where each classifier can deal with the problem correctly. A combination of the results can then be used to recreate the complex decision boundary as presented in Figure 2.11b.

There are two main differentiation methods of ensemble methods; *boosting* and *bagging*. Briefly explained, boosting works by training a combination of learners sequentially. After the learning process, it combines them for prediction, where the later learners focus more on the mistakes of the earlier ones. Bagging (short for bootstrap aggregating), is a parallel ensemble method in which several learners are trained in parallel. The basic motivation in boosting is to exploit the dependency between learners, whereas the motivation in bagging is to exploit the independence between learners [13].

2.3.4. Common Classifier Modeling Algorithms

This section presents standard classifying algorithms in supervised learning used in the course of this master's thesis. There are two main categories in the following lists: single classifiers that only rely on the result that they generate by themselves, and ensemble classifiers that combine the results of single classifiers with the goal of creating a better result. Single classifiers can be categorized into *shallow models* and *deep learning models*.

Shallow models are the traditional machine learning models that have been studied for several decades by now and are considered mature. In addition to the detection effect, they also focus on practical problems, such as detection efficiency and data management. The most common shallow models are listed in the following [10]:

- k-Nearest-Neighbor (kNN): The core idea of this classifier is based on pairwise linear discriminant functions. Its main hypothesis states that if most sample neighbors belong to the same class, then there is a high probability that the unseen sample next to the already-classified sample is the same class. This classification is only carried out to the top-k nearest samples, where k is the number of neighbors considered. Smaller k values increase the risk of overfitting and lead to a more complex model, whereas larger k values lead to simpler models with weaker fitting abilities [10].
- Support Vector Machine (SVM): The strategy of SVMs is to find the maximum separation hyperplane in the available feature space. Only a small number of support vectors can determine this hyperplane; thus, it is a commonly used classifier when there are only small datasets available. However, SVMs can be sensitive to noise near the created hyperplane. The hyperplane separation can be created with the help of linear functions or nonlinear kernel-based functions. Kernel-based functions map the original feature space into a new one so that nonlinear data can be separated [10].
- Quadratic Discriminant Analysis (QDA): This is an adapted form of the standard linear discriminant analysis. Since linear decision boundaries are not appropriate for the many classification problems, QDA can create more complex decision boundaries by using quadratic discriminant functions [12].
- Decision Tree (DT): This tree-like model uses a series of rules for classification. It can automatically exclude irrelevant and redundant features. The training steps of this classifier are feature selection, tree generation, and tree pruning. The tree generation selects the most suitable features and builds child nodes starting from

the root node. Other forms of DT classifier exist, such as random forest, and several boosting and bagging versions. [10].

- Multi Layer Perceptrons (MLP) or Artificial Neural Network (ANN): This classifier relies on a feed-forward neural network to perform the task of classification. They contain a minimum of three layers; an input layer, at least one hidden layer, and an output layer. The units in neighboring layers are fully interconnected and can theoretically approximate arbitrary functions. They therefore promise an excellent fitting ability, especially for nonlinear functions. However, due to the complex model structure, training can be time-consuming. They utilize a technique called backpropagation for training [10].

The classifiers mentioned above all have different strengths and shortcomings; *ensemble methods* try to combine various weak classifiers to implement a classifier with lesser weaknesses. Noticeable classifiers that fall under the category of ensemble methods are the boosting based *AdaBoost* and different bagging based classifiers such as *Random forest*.

Deep learning models play an increasingly important role in the field of IDS. They provide supervised and unsupervised learning models, and since 2015, the number of studies of deep learning-based IDSs skyrocketed, as these models are able to directly learn feature representation from the original data without the need of manual feature engineering. Deep learning models consist of diverse deep networks used to improve the performance of IDSs. When compared with the aforementioned shallow machine learning models, they own stronger generalization and fitting abilities. However, there are also several disadvantages, including *interpretability* and *real-time requirements*. The first problem of interpretability is due to the fact that most deep learning problems are black boxes, that only report the detection results without any interpretable basis. Cyber security decisions should be made cautiously, as decisions taken without an identifiable reason are generally not convincing. The second problem of real-time requirements lies in the fact that deep learning models require longer running times than shallow models and therefore hardly meet the real-time requirements of IDSs. One of the most promising approaches in the field of deep learning is the so-called Convolutional Neural Network (CNN) [10].

These neural networks are designed to mimic the human visual system and obtain significant performance in computer vision, such as object and face recognition. A CNN uses alternating convolutional and pooling layers as illustrated in Figure 2.12. The convolutional layers deal with the feature extracting process and the pooling layers deal with the enhancement of the feature generalizability. Even though CNNs have been recognized as highly accurate in the field of computer vision, they have not been fully exploited in the field of IDS. One reason being that they work on 2-deminsional (2D) input data, therefore the input data for most IDS systems must first be translated into matrices for attack detection. [10, 19].

Even though CNNs are gaining more and more attention, they are not going to be used in this thesis because of several reasons. First, compared to shallow classification models, would have to be translated from 1D data to 2D input data. Furthermore, most of the literature dealing with evolutionary undersampling does not provide results of CNNs,

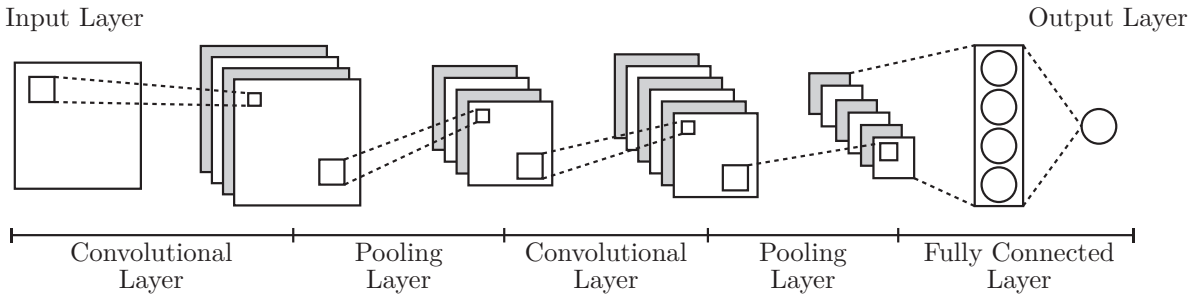


Figure 2.12.: The structure of a CNN, by [10]

therefore making it harder to compare the gained results of the implementation. Another reason is that the authors of [20] also used the same dataset utilized in this thesis and already executed a grid search with different classifiers. Section 4.4 will explain the used dataset and Section 6.1 will then present the used classifiers of the implementation.

Now that the underlying principles of the pattern recognition process have been explained, a major problem needs to be addressed that emerges in many fields in the data science domain; the problem of imbalanced data distribution, as briefly mentioned in Section 2.2. The next chapter will present this problem and explain ways to deal with it. Recent research solutions, specific methods and ways to evaluate the performance of these methods will be explained.

3. The Class Imbalance Problem

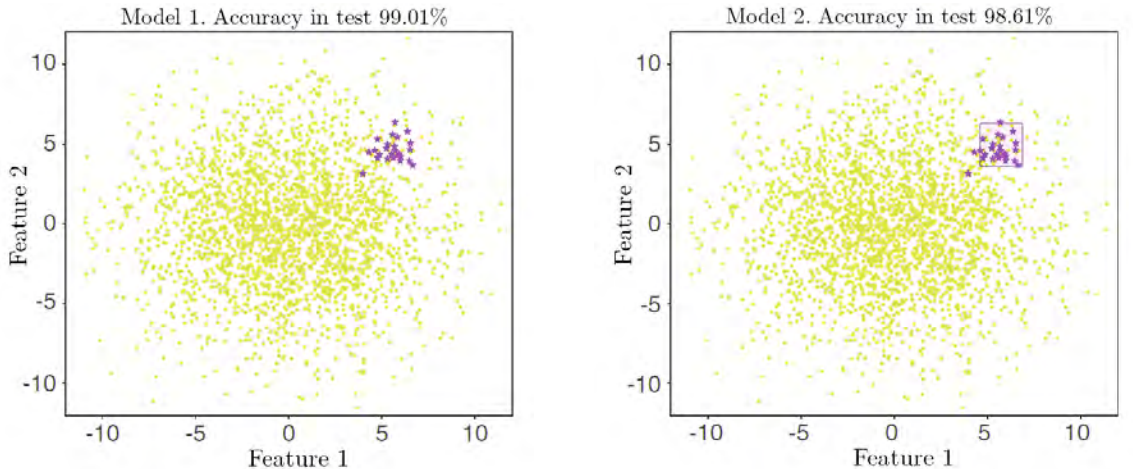
Although the class imbalance problem is a subset of pattern recognition, it is treated as a separate chapter in this thesis because this problem has countless contributions in literature [21–27] dealing with this problem. Therefore, it will be explained in detail below.

The class imbalance problem emerges whenever the classes in a dataset are not evenly distributed. Considering the imbalance of a two-class classification problem would mean that one class is under-represented. This fact usually introduces several difficulties in the learning process of a classifier and worsens the recognition rate of the under-represented class. The under-represented class is usually referred to as the minority or positive class, while the over-represented class is called the majority or negative class. These terms are used interchangeably in the literature; in this thesis, they will be referred to as previously mentioned. In most cases, the minority (positive) class is the most interesting one from a learning perspective [26]. The problem with imbalanced datasets is typically due to standard classifiers that are designed to maximize the accuracy rate. This means that they are trying to classify as many samples as possible in the dataset correctly; therefore, they are biased towards the majority class. Also, as mentioned in Section 2.3.3, classifiers generally try to find a trade-off between performance and simplicity of the classifier. If the imbalance is severe enough, it is easier for the classifier to treat the minority class as noise and ignore them during the learning process, and thus, leading to better performance.

This is an inherent and inevitable problem when using anomaly-based IDS in computer networks. Network intrusions and malicious behavior represents only a small subset of network traffic. Benign traffic generally takes the majority of all generated network packets and thus increase the difficulty of detecting possible attacks. However, their correct detection is crucial for the health of a computer network, as false alarms can lead to loss of relevant packets and drain the focus of administrators [1].

The authors in [25] provide an example considering the "Mammography Dataset" is used, as it has been widely utilized in the analysis of the imbalanced learning problem. This two-class dataset contains images acquired from a series of mammography exams to distinguish if patients are "positive" or "negative" or "cancerous" or "healthy," respectively. Strictly speaking, there are 10923 majority and 260 minority class samples, which leads to an imbalance ratio of 1:37 ($ImbalanceRatio = (MajorityClass)/(MinorityClass)$). In an ideal world, a classifier would provide a balanced degree of predictive accuracy (ideally 100%) for both classes. For the imbalanced mammography dataset, the usual accuracy (3.1) is close to 100% for the majority class and only around 10% for the minority class. This phenomenon is shown in Figure 3.1 from [25]. It has to be noted that both parts

in Figure 3.1 illustrate an imbalanced ratio of 1:100 and use the same dataset to train the classifier. While the left model (3.1a) learned that it should mark each data point as negative (majority), the right model (3.1b) created a decision boundary in the feature space (shown as a purple rectangle). Even though model (a) obtains 0% accuracy of the positive class examples, it still achieves a 99.01% of overall accuracy, due to the fact that all majority examples are classified correctly. On the other hand, model (b) assumes that part of the feature space belongs to the positive class. Since the minority class is the class of interest, this seems to be the most desirable decision. However, this model really obtained 98.61% accuracy, which is 0.4% lower than model (a).



(a) (b)

Figure 3.1.: Examples of two models learned from imbalanced data, from [26]

In the mammography example, an accuracy rate of 10% would lead to around 234 minority samples that are misclassified as majority samples. The consequence of this misclassification is equivalent to 234 cancerous patients diagnosed as noncancerous. In such a delicate problem domain, this result is extremely undesirable. In the medical sector, it can be overwhelmingly costly to misclassify positive samples as negative, and of course, the other way around. The same applies for the field of network anomaly detection. Typically, as mentioned in [1], network intrusions and malicious behavior represent a minimal subset of all network traffic, however, it is highly critical to ensure a balanced accuracy rate between both the majority (normal traffic) and minority (malicious traffic) classes. A high amount of false alarms is unacceptable, as it will lead to loss of relevant packets and frustration of users and administrators. Thus, learning classifiers from unbalanced datasets face a variety of relevant problems, such as data fragmentation, improper inductive bias and noise, absolute or relative lack of data, and improper classification evaluation metrics. Therefore, the main goal with imbalanced data is to find a classifier that provides high accuracy for the minority class without losing too much accuracy in the majority class. As a result, it is also clear that the accuracy of a singular assessment criterion for the performance of a classifier is not sufficient. Fortunately, there are different performance evaluations in imbalanced data that will be discussed in detail in Section 3.3.

The authors of [28] propose that imbalanced data can be categorized into normal imbalanced datasets and highly imbalanced datasets. The first is only slightly unbalanced, meaning that, the records differ at least 10% from perfectly balanced (50-50) datasets, therefore, to count as a normal imbalanced dataset, the distribution of one class must be at least 40% or lower. The second category comprises datasets with a degree of imbalance considered to be extremely unequal. The distribution of highly imbalanced datasets is 10% or lower of one class. Datasets with very high inequality between class distributions are usually accompanied by increased difficulties during the learning process. However, the imbalanced ratio alone is not the only factor that increases the difficulty of the classification problem. For instance, the difficulties in real-world problems could also come from, *small sample sizes*, *overlapping* or *small disjuncts*. These problems are also amplified if the dataset is high-dimensional [26].

The problem of small sample sizes is related to the "lack of density" or "insufficiency of information." It emerges when learning algorithms lack enough data to make useful generalizations about the underlying dataset, resulting in poorly trained models and intensifying in the presence of class imbalance. The majority class may be sufficient to train a reasonable classification model, but the lack of minority class samples increases the difficulty of creating a good generalization. Overfitting is frequently a problem encountered when dealing with small sample sizes [26].

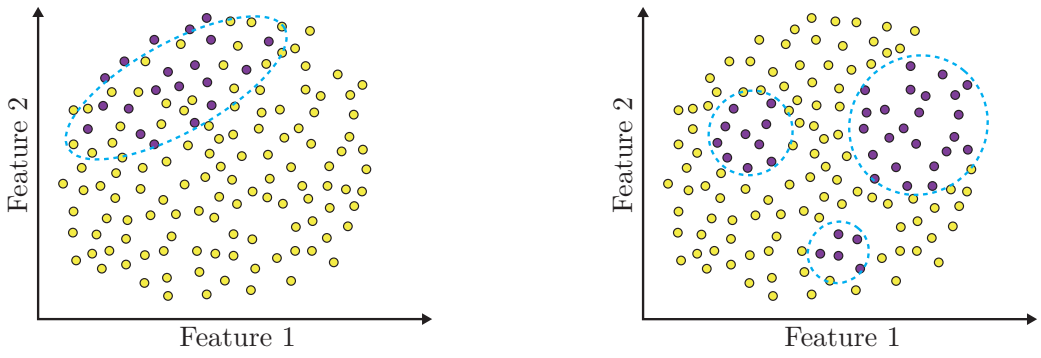


Figure 3.2.: Imbalanced dataset problems (a) class overlapping (b) small disjuncts, from [25]

Data samples in a feature space are referred to as overlapping when examples of both classes are mixed up, illustrated in Figure 3.2a. Simply put, the decision boundary cannot be clearly established to distinguish between both classes. If the data samples overlap, the creation of discriminatory rules becomes more difficult, resulting in more general rules needing to be created that classify a high number of minority class instances. The less overlapping between classes exist, the less problematic the imbalance ratio gets, because even simple classifiers would be able to distinguish between classes without overlapping.

The last problem, called small disjuncts, illustrated in Figure 3.2b, is the problem of the minority class samples forming multiple fields. In the literature [26], this problem is referred to as subconcepts being formed within the concept of the minority class. Those

smaller subconcepts tend to increase the complexity of decision boundaries because the number of instances is generally unbalanced.

Two lessons can be learned from the previous mentioned drawbacks to imbalanced data distributions. First, accuracy is no longer a good measure of correctly classified examples of distinct classes. It can lead to erroneous conclusions, as the overall accuracy can still be around 90%, in a dataset with an imbalance ratio of 1:9, when all classes are classified as negatives. Therefore, more informative measures in this context are required, such as ROC, geometric mean, f-measure, precision or recall, which will all be explained thoroughly in Section 3.3. The second conclusion is that there needs to be a solution to construct classifiers that are biased towards the minority class without being too harmful to the accuracy of the opposing class [26]. The next part of this thesis will deal with solutions that can be categorized into four groups.

3.1. State-of-the-art Approaches

A wide range of research has addressed different approaches to deal with the unbalanced class distribution in datasets. Publications such as [2, 26–28] and many more have addressed this problem. Generally, the techniques are separated depending on how they deal with the problem. Those categories are: *Data level approaches, algorithm level approaches, cost-sensitive learning approaches,* and *ensemble-based approaches*. In the following section, these are all briefly discussed along with notable methods for each category, as some of them may be used during the implementation phase of this thesis. Figure 3.3 gives an overview of the approaches mentioned above and shows that solutions by cost-sensitive learning include both methods of data-level and algorithm-level approaches. Ensemble-based approaches are referred to as boosting within the data-level approaches as they use a combination of data level learners.

3.1.1. Data Level

Data level approaches, also called external approaches, aim at balancing the class distribution by adjusting both classes to the same level. This eliminates the classifier bias towards the negative class as the imbalance in the dataset is removed or reduced. Thus, this approach is overall more versatile because no modification to the classifiers has to be made. Rebalancing the class distribution is accomplished by using the preprocessing methods over- and undersampling. *Undersampling* focuses on reducing the majority class to equalize the number of examples in each class, and *oversampling* generates or replicates samples from the minority class in order to reduce or eliminate the imbalance ratio [26].

As this thesis focuses mostly on resampling methods, a detailed explanation to over- and undersampling and selected methods can be found in Section 3.2.

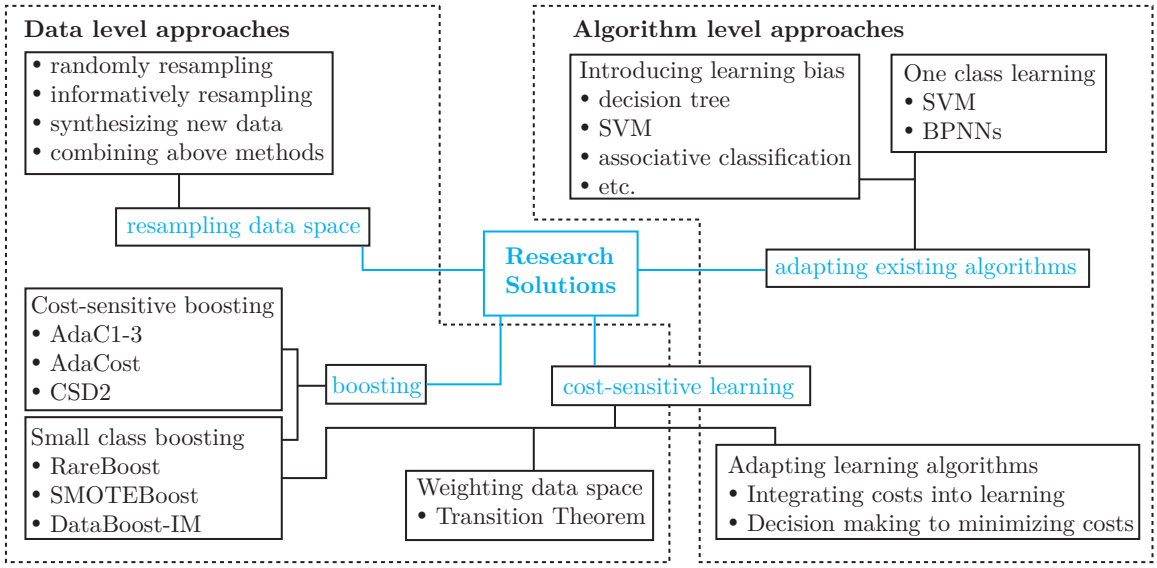


Figure 3.3.: Research solutions to the problem of classification of imbalanced data, from [29]

3.1.2. Algorithm Level

Algorithm level approaches, also referred to as internal approaches, try to adapt existing classifier learning algorithms to improve the imbalanced class problem by biasing the learning procedure towards the minority class. The major drawback is that they require specialized knowledge of the classifier and the application domain. This knowledge is necessary because these solutions try to comprehend why the classifier is failing when the distribution of classes is imbalanced [26]. Generally, there are three different categories: one-class learning, adapting learning algorithms, and choosing an appropriate inductive bias. Approaches for finding the right balance of bias are, for example, probabilistic estimates at the tree leaves, or development of new pruning techniques for decision trees. For SVMs, solutions include using different penalty constants for different classes or are adjusting the class boundaries on a kernel-based alignment ideal. Adapting learning algorithms can be done by integrating the costs into the learning process and tweaking the decision-making process to minimize costs. One-class learning is a system where only one class is used to make boundaries that surround the target concept; algorithms that implement this type of learning are neural networks and SVMs [29].

3.1.3. Cost-sensitive Learning

This approach is a combination of algorithm and data level approaches, as it needs both data-level transformation and algorithm level modifications. The data level transformation is needed to add the costs of misclassification to each instance. Further, algorithm level modification is necessary to accept those misclassification costs in the classifier learning algorithm. By penalizing mistakes on the minority class, this approach tries to bias the classification results away from the majority class. A significant drawback of this

approach is that the costs need to be assigned to each class by either a domain expert or an estimate using training data [26].

As depicted in Figure 3.3, cost-sensitive learning approaches can be distinguished by three main categories: data space weighting, making a specific classifier learning algorithm cost-sensitive, and using the Bayes' risk theory to assign each sample to its lowest risk class. Weighting the data space deals with modifying the distribution of the training set with regards to misclassification costs such that it is biased towards the costly classes based on the theoretical foundations of the Translation Theorem. The second category makes a specific classifier learning algorithm cost-sensitive. For example, in decision trees, the tree building strategies can be adapted for minimizing misclassification costs. Cost-information can either be used to choose the best attribute to split the data or to determine if a subtree should be pruned. For the last category, the Bayes' risk theory is used to assign each sample to its lowest risk class. A typical decision tree, for example, assigns each class label on a leaf node depending on the majority class that reaches the node. With cost-sensitive algorithms, the class label is assigned to the node that minimizes the classification cost [29].

3.1.4. Ensemble-based

The results of ensemble-based approaches have recently shown promising results in the domain of class imbalance [30]. They are usually created by combining an ensemble-based learning algorithm (Section 2.3.3) and the previously mentioned data level and cost-sensitive approaches. Data level approaches are used by applying an over- or under-sampling preprocessing algorithm before the ensemble approach is trained. Cost-sensitive ensembles include costs to either each base classifier or globally to the whole ensemble.

3.2. Resampling to Combat Class Imbalance

As mentioned above, over- and undersampling techniques pursue the same ultimate goal. They try to balance a given dataset, but there are significant differences that lead to advantages and disadvantages. Undersampling techniques only change the majority class by reducing the number of majority class items, thus balancing the imbalance ratio. This method results in a smaller dataset, which could be a problem if the dataset is already relatively small at the beginning. Oversampling techniques, on the other hand, only change the minority class by generating new class items, leading to a generally larger dataset that balances both majority and minority class [31].

Since this thesis is trying to show that evolutionary undersampling methods can improve intrusion detection systems' performance, the main focus will deal with data level techniques and, in particular, undersampling methods. However, to give a brief overview of oversampling methods as well, the most common ones will also be explained.

3.2.1. Undersampling

Within this thesis, only undersampling is taken into consideration because various research [2, 22, 28, 30] has shown that undersampling is able to deliver promising results and help to reduce the need of resources due to the smaller sample size in the resulting undersampled dataset.

The first method is called ClusterCentroids, which undersamples datasets by replacing the original samples using an algorithm called clustering using representatives (CURE). This algorithm identifies clusters (e.g. centers of classes) and replaces the relevant points within these clusters while maintaining the underlying cluster structure. ClusterCentroids differs from other undersampling methods in the way that it generates samples in the majority class. Other methods only use existing majority class samples for their resampling process [32].

The following figures in this Section 3.2 are created using the *imbalanced-learn* toolkit [33]. Each figure contains 250 samples with 28 samples in the minority class and 222 in the majority class. This results in an imbalance ratio of roughly 1:8. Purple points represent the minority class samples, and yellow points represent the majority class, respectively. Slightly transparent data points represent the samples that were removed by the undersampling method. Figure 3.4 shows an example of ClusterCentroids undersampling.

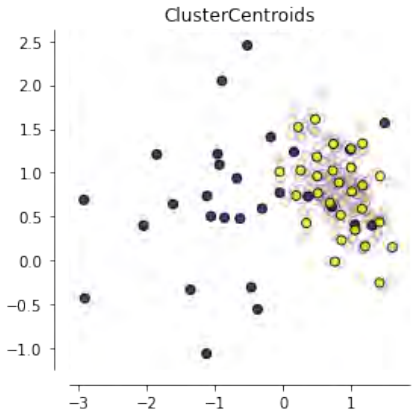


Figure 3.4.: Undersampling example using ClusterCentroids, by [33]

The following undersampling methods select samples from the existing majority class samples. This type of undersampling can be divided into two groups: *controlled* and *cleaning*. While controlled methods are able to manage the number of majority samples taken, cleaning methods only remove specific samples that are considered unnecessary without specifying the exact number [32].

RandomUnderampler (RUS) is the simplest and simulataneosly, the riskiest way to make such a selection by randomly selecting samples from the majority class. It is risky because it deletes random samples without checking their potential significance or relevance. An example is shown in Figure 3.5.

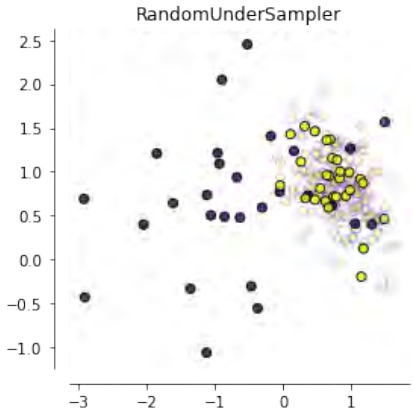


Figure 3.5.: Undersampling example using RandomUndersampling, by [33]

There are three different versions of the NearMiss algorithm, and all versions implement some heuristic rules to select samples. NearMiss-1 selects samples from the majority class, where the average distance to the k nearest samples of the minority class is the smallest. NearMiss-2 selects majority class samples with the minimum average distance to the farthest minority class samples. Finally, NearMiss-3 consists of two steps; first, the m nearest-neighbors for each minority sample will be kept, then the majority class samples with the maximum average distance to the k nearest neighbors is the largest are selected [32]. All three methods are shown in Figure 3.6.

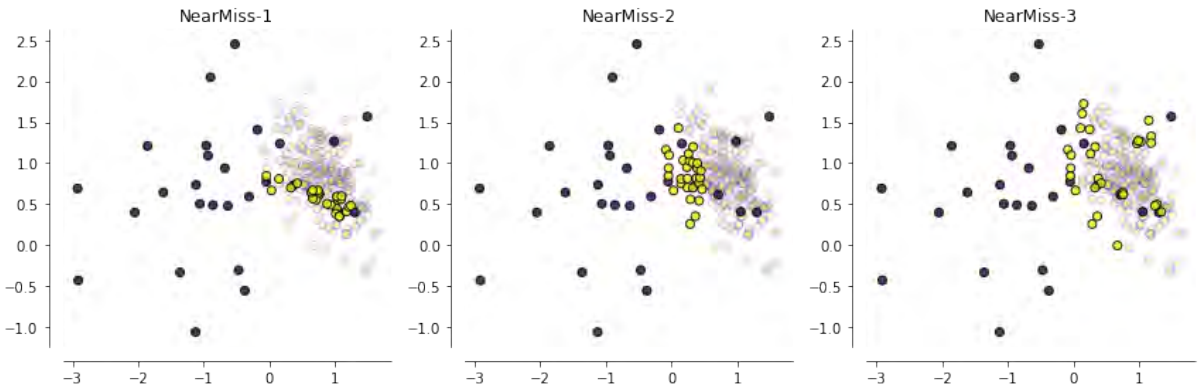


Figure 3.6.: Undersampling example using different NearMiss versions, by [33]

EditedNearestNeighbor (ENN) edits the dataset by removing samples of the majority class that do not "agree enough" with their nearest neighbors. A strict option of "agreeing" in this case could mean that all neighbors who are not the same class are removed. It is possible to run this selection process multiple times iteratively, which would lead to the principle of RepeatedEditedNearestNeighbor (RENN). AllK-Nearest-Neighbor (AllKNN) is only slightly different from RENN; instead of repeating the same process repeatedly, AllKNN increases its k parameter of its internal nearest-neighbor algorithm each iteration [34]. Figure 3.7 shows these relatively similar undersampling approaches.

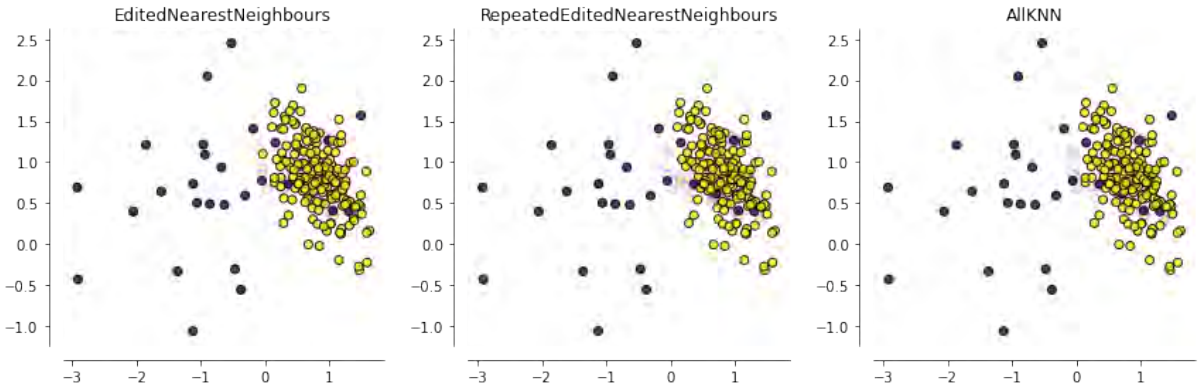


Figure 3.7.: Undersampling example using ENN, RENN and AllKNN, by [33]

TomekLinks (TL) forms links by connecting two nearest neighbor samples from different classes with each other. There are two variants of this undersampling method; first, there is the option to remove both samples that form a link, and the other is to only remove the majority class sample in this link. The different modes are presented in Figure 3.8.

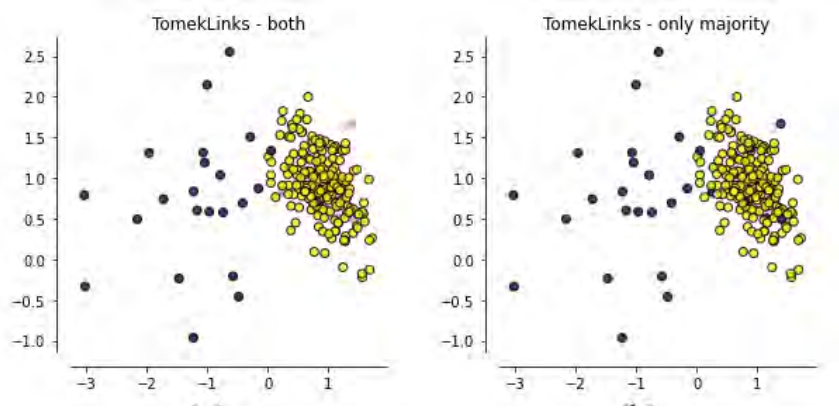


Figure 3.8.: Undersampling example using TomekLinks, by [33]

Condensed-NearestNeighbor (C-NN) is using the one nearest neighbors algorithm to choose which majority sample can be removed. The issue with this method is that it is sensitive to noise by preserving noisy samples. OneSidedSelection (OSS) adds the use of TomeLinks to C-NN to remove links that are considered noisy. NeighborhoodCleaningRule (NCR) combines C-NN and OSS and additionally uses the previous mentioned ENN algorithm to remove more noise samples. After that, misclassified instances are removed by using the 3NN rule [32]. The differences can be seen in Figure 3.9.

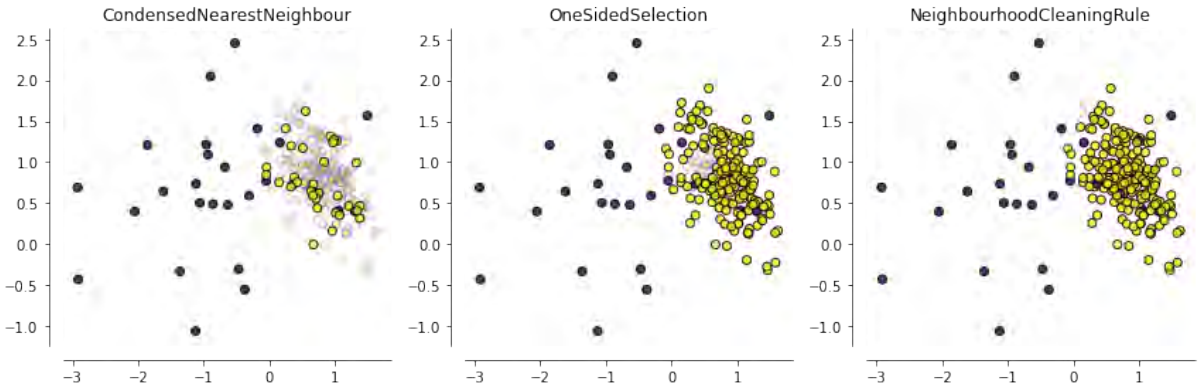


Figure 3.9.: Undersampling example using C-NN, OSS, NCR, by [33]

3.2.2. Oversampling

For completeness, some popular oversampling methods will be discussed in this section. The first method is RandomOversampling; it randomly replicates data samples until the classes are balanced. The results of this method can be seen in Figure 3.10, the darker purple data samples on the right side of the Figure indicate that the replicates overlap with the original minority samples [26].

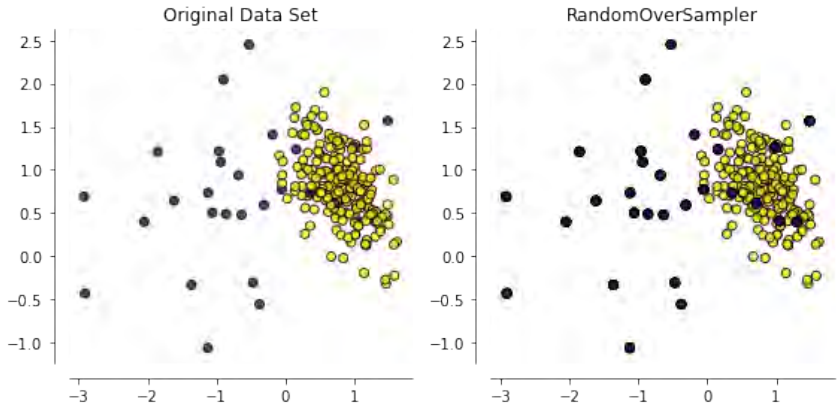


Figure 3.10.: Oversampling example using RandomOversampling, by [33]

Instead of simply replication data samples, Synthetic Minority Oversampling TEchnique (SMOTE) tries to introduce synthetic examples by interpolating between several minority samples that lie together. As examples, at the border and boundary are more likely to be misclassified, SMOTE-Borderline was introduced to create new synthetic examples along with these borderline instances. SMOTE-SVM instead, focuses on the creation of samples on the decision borders of minority and majority classes created by the SVM classifier [26]. A comparison of these SMOTE variants is illustrated in Figure 3.11.

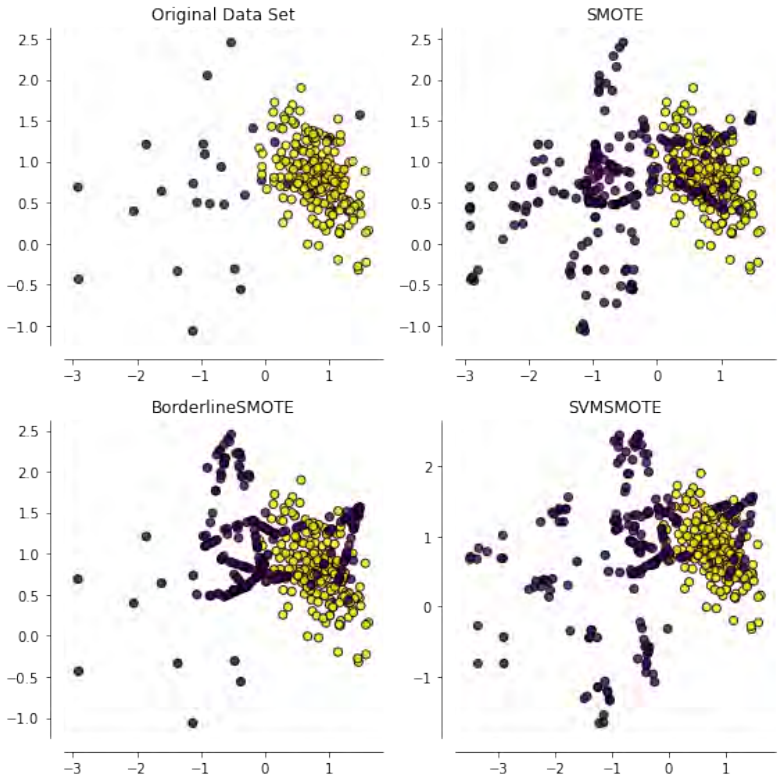


Figure 3.11.: Oversampling example using different SMOTE variants, by [33]

SMOTE has been the inspiration for almost all the conventional oversampling methods used within the imbalanced class problem. The next algorithm, Adaptive Synthetic (ADASYN) oversampling, also uses SMOTE and has two main goals: first to reduce the bias towards the majority class and then to shift the decision boundaries to harder classifiable examples. ADASYN calculates the k nearest-neighbor from the majority class for each minority example. After that, it decides how many examples should be generated for each majority based on a weighting algorithm. The last step is the crucial difference compared to SMOTE, because of the equal number of newly generated samples for each minority example [31]. Figure 3.12 shows an example of this algorithm.

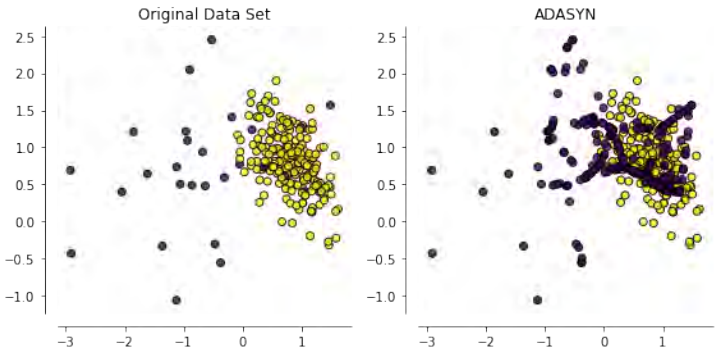


Figure 3.12.: Oversampling example using ADASYN, by [33]

3.3. Performance Analysis

Analyzing the performance of classifiers and learning algorithms is a challenging task under the presence of class imbalance. As mentioned before, measures such as accuracy may mask an inferior classification performance if the imbalance ratio is too high. Naturally, it is of utmost importance to alleviate this problem by choosing suitable measures in infrequent classes. This section presents different performance measures used in classification and elaborates on additional steps necessary to obtain appropriate results. In Section 2.3.3, the topic of *data splitting* has already been briefly introduced, following the fundamental importance in classification and in unbalanced dataset classification is dealt with in more detail.

When dealing with classification problems, it is often helpful to gain a common range of values within the dataset, in which a method called *scaling* is used as a preprocessing procedure. There are three common scaling or normalizing methods: interval fit, z-score scaling (or standardization), and arctangent scaling. The interval fit scaling method transforms the available data to fit into a certain interval, commonly [0,1]. Z-score, on the other hand, scales the features in a way that that they have the properties of a standard normal distribution with a mean of zero and a standard deviation of one. Arctangent scaling will fit the available data within the interval [-1,1], thus ensuring that features near the mean are scaled almost linearly and outliers are still within the interval range. All those methods can lead to easier processing and distinction of patterns because of smaller variability within the features [16, 17].

The usual way to obtain training and test sets is by splitting a given set into two parts. When splitting, it is desired to keep all the defining properties of the original dataset as much as possible. If the properties are ignored, the chances increase that the validation set would provide misleading estimates. For instance, the training set might only include the majority class, while the test set only contains majority instances. In order to conquer this problem, a common method called *stratified sampling* is used. In stratified sampling, a split percentage can be chosen to split the data accordingly, and this percentage is maintained for both training and test sets. Sometimes there is not enough labeled data available to create a proper test dataset. If this is the case, a validation method called cross-validation can be used. With k-fold cross-validation, the original dataset is divided into k ($1 \leq k \leq n$) equal subsets by stratified splitting. Then $k-1$ subsets are used to train the classifier while the remaining subset is used for performance evaluation. This process is then repeated k times so that each subset is used once for testing. The result of the cross-validation is then the average of the k runs. Usually, this k-fold cross-validation is repeated multiple times to reduce the influence of randomness introduced by the data split. Common configurations include 10-times 5-fold cross-validation and 5-times 2-fold cross-validation. In an extreme case, if the number of folds equals the number of samples in the original dataset, there is only one instance in each test set. This method is called *leave-one-out* validation [13].

When evaluating classifiers' performance, it is key to properly assess its quality with respect to other classifiers dealing with the same problem. As mentioned in Section 2.3.2, the results of correctly and incorrectly classified examples of each class can be stored in

a confusion matrix (Table 3.1).

To evaluate the performance of classifiers, historically, the accuracy (ACC) (3.1) has been the most commonly used measure [2, 30]. With imbalanced class distributions, however, it is not a suitable option because accuracy rate weights the influence of classes concerning their number of instances in the dataset. This means that classes with more instances have a greater influence on the accuracy rating, which is not desirable in an imbalanced scenario.

$$Acc = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{FN} + \mathrm{FP} + \mathrm{TN}} \tag{3.1}$$

	Positive prediction	**Negative prediction**
Positive class	True positive (TP)	False negative (FN)
Negative class	False positive (FP)	True negative (TN)

Table 3.1.: Confusion matrix for a two-class problem, from [30]

As this thesis deals with imbalanced domains, other measures that calculate the performance of each class independently without being influenced by the other class need to be considered. By using the confusion matrix (Table 3.1), different measures can be calculated to evaluate the performance of each class independently. Four metrics are especially interesting: the false negative rate (FNR) (3.5), false positive rate (FPR)(3.4), true negative rate (TNR) (3.3), and the true positive rate (TPR) (3.2). On the one hand, there is FNR and FPR that represent the percentage of cases misclassified. On the other hand, TNR and TPR represent the cases that are correctly classified [28].

$$\textit{True positive rate (also known as Sensitivity or Recall)}\ \mathrm{TP}_{rate} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{3.2}$$

$$\textit{True negative rate (also known as Specificity)}\ \mathrm{TN}_{rate} = \frac{\mathrm{TN}}{\mathrm{FP} + \mathrm{TN}} \tag{3.3}$$

$$\textit{False positive rate}\ \mathrm{FP}_{rate} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} = 1 - \mathit{Specificity} \tag{3.4}$$

$$\textit{False negative rate}\ \mathrm{FN}_{rate} = \frac{\mathrm{FN}}{\mathrm{TP} + \mathrm{FN}} = 1 - \mathit{Recall} \tag{3.5}$$

However, these measures consider only one of the classes when used on their own. For this reason, there are two commonly considered measures to use in this scenario, the geometric mean (GM) (3.6) and the Area Under the ROC Curve (AUC) (3.7). The GM balances the accuracy between majority and minority instances while at the same time,

being able to deal with the class imbalance problem. A poor performance in prediction of the positive examples will lead to a low GM value, even if the negative examples are correctly classified.

$$\mathrm{GM} = \sqrt{\mathrm{TP}_{rate} \cdot \mathrm{TN}_{rate}} \tag{3.6}$$

The AUC, on the other hand, is computed as the area under the Receiver Operating Characteristics (ROC) curve. The ROC curve is created by plotting the TPR against the FPR at different classification thresholds. To put in another way, it plots the false alarm rate versus the hit rate. It visualizes the trade-off between TPR (benefits) and the FPR (costs), showing that it is not possible for any classifier, to increase the number of true positives without increasing the number of false positives, respectively.

Therefore, the AUC corresponds to the probability that the model ranks a random positive example more highly than a random negative example. AUC ranges in values from 0 to 1. A model where all predictions are wrong has an AUC of 0.0; one where all predictions are 100% correct has an AUC of 1.0. It is widely used in imbalanced domains [28].

$$\mathrm{AUC} = \frac{1 + \mathrm{TP}_{rate} - \mathrm{FP}_{rate}}{2} \tag{3.7}$$

Another commonly used metric in imbalanced classification is the F-Measure (3.9). It focuses on analyzing the positive class and tries to find the trade-offs between correctness and coverage in classifying. For its calculation, it uses a weighted harmonic mean between precision (3.8) and recall (3.2). Precision measures how many of all predicted minority class samples are classified correctly, whereas, the recall shows how many instances of all real minority samples are predicted correctly. The F-measure combines these performance metrics and adds a weighting factor, β, to influence the importance given to each term. Usually, $\beta = 1$ is used leading to the F_1 measure, shown in (3.10) [26].

$$\mathrm{Precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{3.8}$$

$$\mathrm{F}_\beta = \frac{(1 + \beta^2) \cdot \mathrm{Recall} \cdot \mathrm{Precision}}{(\beta^2 \cdot \mathrm{Precision}) + \mathrm{Recall}} \tag{3.9}$$

$$\mathrm{F}_1 = 2\frac{\mathrm{Precision} \cdot \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}} \tag{3.10}$$

4. Methodology

After the introduction into the class imbalance problem and the ways to overcome it, this chapter gives insight into the methods used in this thesis to tackle the problem in network anomaly detection. First, evolutionary algorithms are explained in Section 4.1. A modern field of the Evolutionary Algorithm (EA) is the Genetic Algorithm (GA), presented in Section 4.1.1. These algorithms are the cornerstone of evolutionary undersampling, as their methods are used to create a supposedly superior sampling method. EUS, its taxonomy, and the concept behind it are explained in Section 4.2. Furthermore, methods to evaluate the performance of undersampling methods compared to other sampling methods will be discussed in Section 4.3. Finishing this chapter, the used dataset and its structure, quantities, and detailed description will be presented in Section 4.4. Alltogether, this chapter deals with the used methods and resources to prepare for the implementation in Chapter 5.

4.1. Evolutionary Algorithms

Evolutionary algorithms are, in fact, not a very new research topic; early computing pioneers such as Alan Turing and John von Neumann already discovered the possibilities of these algorithms. They formed ideas around machine learning, biological mathematics, and biological automation using evolutionary approaches. These visionaries realized that exploitative methods could only go so far to solve a set of complex problems; that is why they focused on more exploratory methods. Compared to exploitative approaches that focus more on direct local knowledge to generate a solution, an EA takes a more exploratory or stochastic approach (leaping into the unknown). These algorithms form a group of biologically-inspired algorithms that take advantage of synthetic methods, such as management of populations, replication, variability, and selection. These methods all rely on the fundamental theory of Darwinian evolution in that they produce a simulation in which the survival of the fittest creature leads to passing on their genes to the next generation. In general, these algorithms are often simple at the higher level, but they become very complex the more domain knowledge is brought into the system. An EA generally needs some form of quantitative goal to work; this goal is the metric that can determine success or failure. The success metric can also be used to end the algorithm early if some threshold is reached [35].

Evolution can be viewed in two ways; the first is a biological view dealing with various interactions of biological systems, and the other is from the computer science perspective. This thesis deals with the computer science view, but mixed with a hint of biology to

better visualize the underlying principles. Evolution always includes a population of entities (potential solutions), where some form of *replication*, *variation* and *selection* occurs dynamically. The three dynamic mechanisms can be shown in Figure 4.1, they represent a stochastic but guided process where the main goal is to move towards a fixed goal.

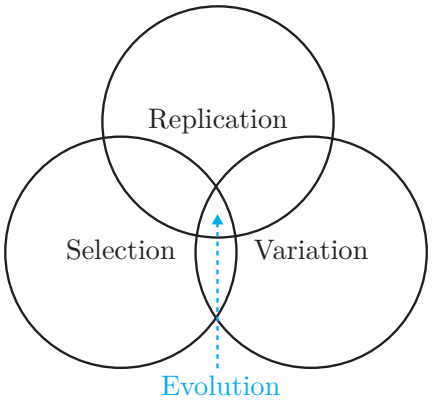


Figure 4.1.: Idealized Darwinian evolution, by [35]

Replication is usually achieved by forming new entities either through creating an entirely new generation of the population or altering an existing generation, whereas *variation* is commonly achieved by either crossover (also called recombination) or mutation. Crossover creates new solutions by combining parts of other entities, while mutations introduce randomness into the population by randomly changing features of solutions. Finally, *selection* is based on Darwin's theory of natural selection or survival of the fittest. Within this subpart, entities should be selected that show the most promising solutions and are thus carried forward to create children for the next generation [35].

Evolution in computer science, also called digital evolution, is a development in which a population of entities undergoes several generational changes. Every one of these changes starts with a selection from the previous generation. To select an entity, an evaluation process checks each entity against the known goal. After the selection, some replication occurs. Each replication process performs some form of variation, by either recombination or mutation. Figure 4.2 shows this necessary digital process that is adopted by many EAs. This high-level evolutionary process, as depicted, seems to be a relatively simple and straightforward procedure, but within this process, various complex nuances and variations appear. These variations are mainly influenced by evolutionary biology [35]. The general ideas of these steps can be carried over to the next section of genetic algorithms. The goal to reach the best possible solution will be explained in detail.

4.1.1. Genetic Algorithms

These algorithms are inspired by the Darwinian theory of evolution [36], a theory where the survival of the fittest creature and their genes are simulated. Its main inspiration is the natural selection of the fittest individuals for reproduction so that their offspring inherit the characteristics of their parents and forms the population of the next generation.

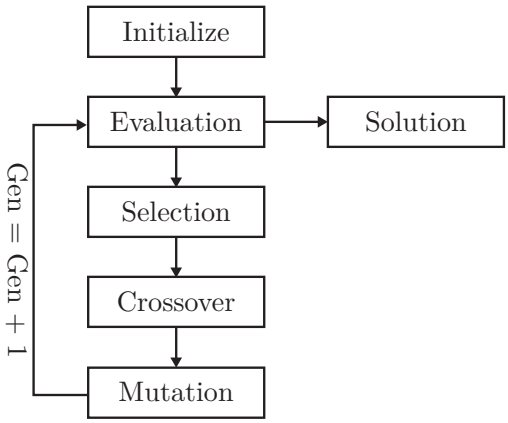


Figure 4.2.: General steps in evolutionary algorithms, by [35]

A Genetic Algorithm (GA) evaluates the fitness of each individual in the population by using a fitness function. If their parents have scored a high fitness value, their offspring will likely develop good genes and thus have a better chance of surviving. This process keeps iterating, and at the end, a generation with the best-suited solution to the problem will be found [35].

This algorithm begins with a set of individuals that is called *population*. Each individual represents a solution to a problem and is called *chromosome*. Chromosomes are characterized by a set of parameters (features or variables) known as *genes*. Figure 4.3 illustrates the different parts of a GA. In the case of an undersampling problem, a gene could be one sample of the majority class, and the chromosome could represent the set of reduced samples from the majority class. The population would consist of different versions of the undersampled majority class sets.

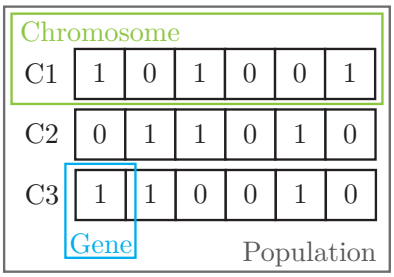


Figure 4.3.: Parts of a genetic algorithm, created from [37]

As GAs have a stochastic background, their reliability might be questioned. Even with several randomized steps, the algorithm is able to stay reliable by estimating the global optimum for a given problem. This process maintains the best solutions in each generation and uses them to improve the upcoming solutions, thus improving the entire population generation by generation. Creating new solutions in the crossover process by combining them results in exploiting the area between the chosen parent solutions. Diversity can be promoted by implementing mutation and, therefore even further increasing the exploratory behavior of the GA. Mutation might randomly create substantially better solutions and ultimately lead to other solutions towards the global optimum [37].

As mentioned earlier, most of the evolutionary algorithms build on the steps illustrated in Figure 4.2. The same concepts exist with GA; these steps are explained in the following [37]:

1. **Create the population**: The initial population includes multiple solutions (chromosomes) to a problem. This step's primary focus is to spread the solutions in the search space as uniformly as possible. A diverse set of solutions increases the chance of finding promising regions.
2. **Determine fitness**: To determine how to fit a solution is (i.e., the ability to compete with other solutions), a fitness value needs to be calculated. The probability that a solution will be selected for reproduction in the next step is based on the fitness value. Calculations for the entire population can be a time-consuming activity, depending on the complexity of the fitness function and population size. This function is why EAs are highly adaptive. Each method to obtain a fitness value is problem-dependent; for instance, in the pattern selection domain, a fitness function could be the accuracy of a chromosome.
3. **Select the mating pool**: The main inspiration for this component is natural selection. In nature, the fittest individuals exceed others with higher chances of obtaining food and mating. Therefore, their genes are more likely to contribute to the production of newer generations. As GAs are inspired by that idea, a roulette wheel selection operator is used. This method assigns every solution a probability proportional to their fitness value. Figure 4.4 illustrates an example for 6 different individuals. Details of each individual are presented in Table 4.1. In these examples, Individual 5 is the fittest solution with the largest probability to be selected for the next generation. As the roulette wheel method is a stochastic operator, poor solutions have a small chance of participating in the next generation. Using "lucky" poor individuals will increase the diversity of the population.

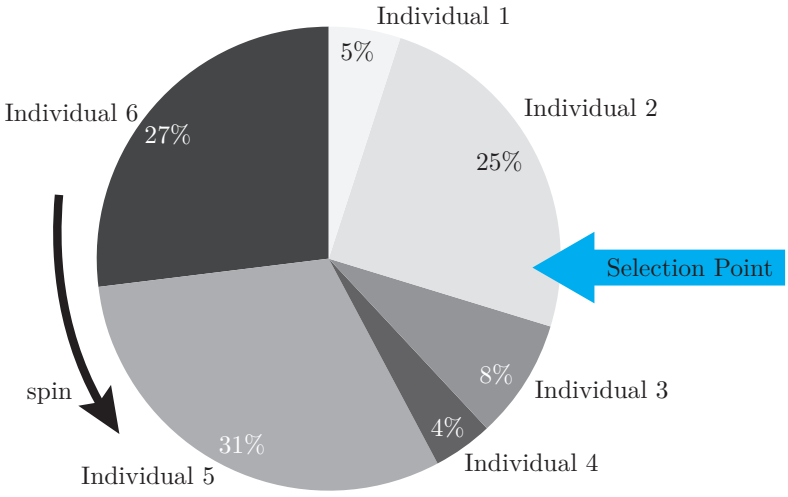


Figure 4.4.: Mechanism of the roulette wheel in a GA, by [37]

Individual Number	Fitness value	% of Total
1	12	5
2	55	24
3	20	8
4	10	4
5	70	30
6	60	26
Total	227	100

Table 4.1.: Details of individuals in a roulette wheel, by [37]

4. **Breed new generation**: In nature, after the individuals for reproduction are selected, the genes of a male and female are combined to produce new chromosomes. In GAs, this is simulated by combining two individuals, also called parent solutions, selected by the roulette wheel. Different techniques for this crossover exist, and commonly used are two methods: single-point and double-point, as illustrated in Figure 4.5. With single-point crossover, a randomly chosen index decides which part of each parent is used by splitting at this point and swapping each parent's genes. In a double-point crossover, two randomly chosen crossover points mark an area that is swapped.

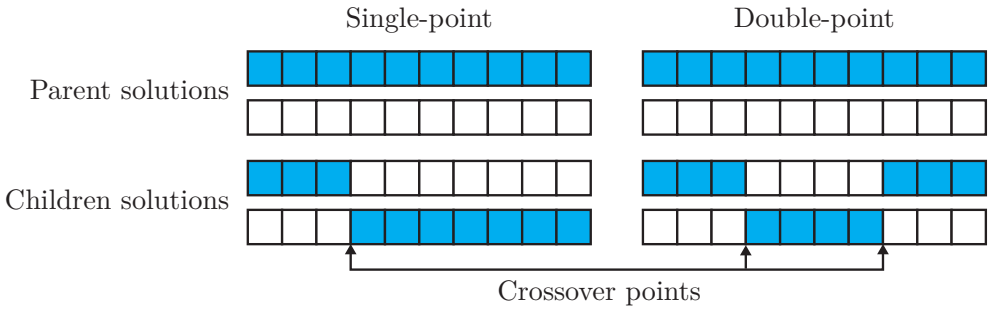


Figure 4.5.: Single-point and double-point crossover techniques, by [37]

5. **Mutate new generation**: The last component of a GA maintains the diversity in the population and plays a key role in the exploration of the search space by changing multiple genes in new solutions. This relatively low mutation rate prevents the solutions to become too similar. Figure 4.6 visualizes randomly selected genes to be changed.

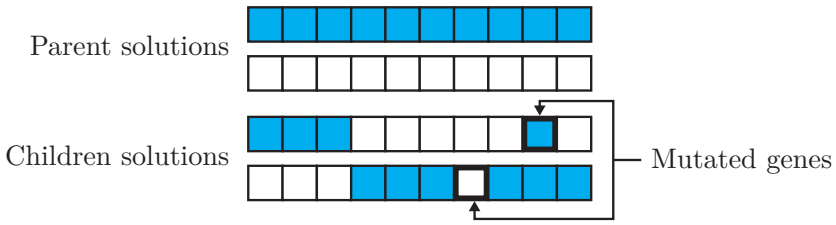


Figure 4.6.: Mutation operator in a GA, by [37]

GAs start with a random population and then loop through steps 2-5, as they represent the evolutionary operators (selection, crossover, and mutation), to improve the quality of the genes over each iteration. Another commonly used evolutionary method to further improve quality is called *elitism*. With this added method, one or multiple best solutions of the previous generation are maintained and transferred to the next generation without modification. This method's main idea is to keep a good solution without the chance of degrading them when applying crossover and mutation components [37].

With the methods explained above, an undersampling method can be created, which will be the focus of the next section.

4.2. Evolutionary Undersampling

The undersampling method used in this thesis relies on the principles of the GA mentioned above. EUS has already shown its usefulness in real-world applications [38]: it is an evolutionary prototype selection algorithm adapted to work within the class imbalance problem. Prototype selection is a sampling procedure that aims to reduce the reference set for the nearest neighbor classifier. The reduced reference set helps to improve its accuracy and reduces the amount of storage needed. However, the general objective of prototype selection and undersampling is different. The balance of the data distribution is more critical in an imbalanced scenario than reducing the overall size of the dataset. For this reason, EUS tries finding a useful undersampled dataset where the search for this solution is guided by a GA [2].

In [28] the taxonomy of EUS is introduced; consisting of a total of eight EUS methods. The first differentiation can be done by the objective that EUS pursues and how it selects instances. Concerning the objective, the authors suggest two goals: Evolutionary Balancing Undersampling (EBUS) aims to balance the data without lacking effectiveness in classification accuracy, and Evolutionary Undersampling guided by Classification Measures (EUSCM) aims to achieve the optimal power of classification without considering the balancing of the data. Regarding the types of instance selection, there is Global Selection (GS) that also allows minority and majority examples to be removed, and Majority Selection (MS) that only allows removal of majority classes. This categorization of EUS produces four subgroups that can further be separated by the measure of evaluation; these measures are AUC and GM. Figure 4.7 shows the full taxonomy introduced in [28].

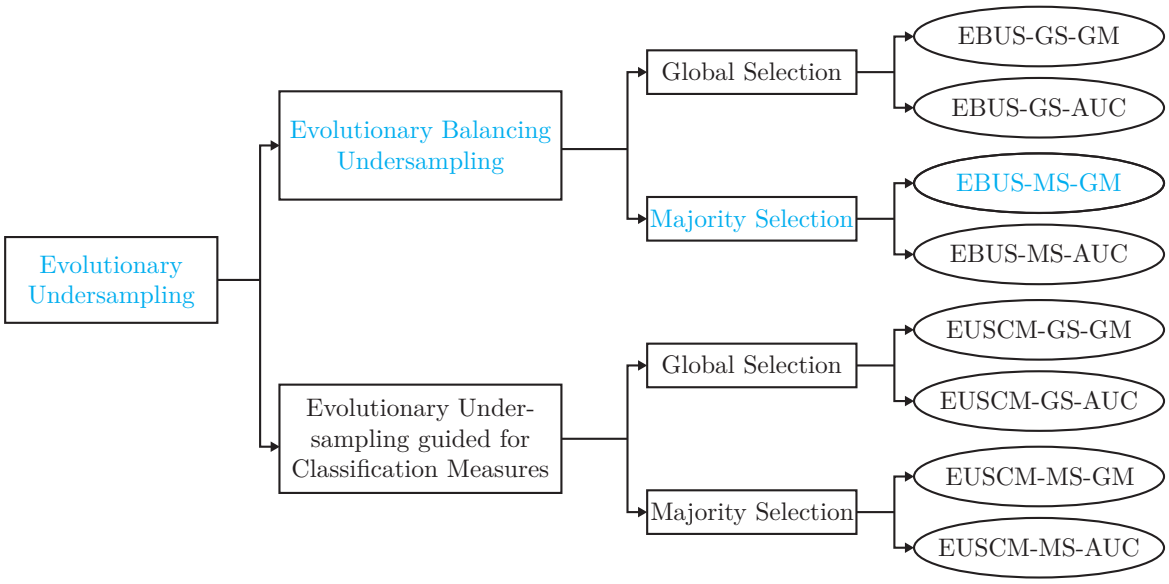


Figure 4.7.: Evolutionary undersampling taxonomy, by [28]

As mentioned in Chapter 3, network traffic is usually prone to very high imbalance ratios. The authors of [28] achieved the most promising results with EBUS-MS-GM when using highly imbalanced datasets; therefore EBUS-MS-GM is considered and explained in this thesis. The blue highlights in Figure 4.7 illustrates this path.

Initially, EUS creates a set of randomly undersampled data subsets, which are then evolved by the steps of selection, crossover, and mutation until the best undersampled dataset cannot be further improved in terms of the fitness function. In EUS, chromosomes are represented as a binary vector, where each gene either represents the absence or presence of the corresponding instance in the dataset. EBUS-MS only considers the majority class for removal, so all majority class examples are included in each chromosome but flagged as 1 or 0, respectively. Minority class instances are automatically added to the final dataset; in other words, a binary one tells the algorithm that this instance is included in the solution, and a binary zero means that this sample is not used for this chromosome. Each chromosome is represented as follows [28]:

$$V = (\nu_{x1}, \nu_{x2}, \nu_{x3}, \nu_{x4}, ..., \nu_{x_{n^-}}) \tag{4.1}$$

where ν_{x_i} is either set to 0 or 1 and n^- represents the number of majority class instances.

In order to estimate the performance, a fitness function is used that considers the balancing between the minority and majority class instances. EBUS-MS-GM uses the GM as a performance measure, maximizing the accuracy of both classes at the same time. The authors of [30] use a 1NN classifier combined with a leave-one-out cross-fold validation to calculate this fitness value. As this calculation has to be done for every generation, this process can be very time-consuming, hence the 1NN classifier is a resource-friendly

and fast choice for this step. The fitness function proposed by [30] provides a good trade-off between data balancing, reduction and accuracy in classification. Finally, the fitness function is defined as followed:

$$\text{fitness}_{\text{EUS}} = \begin{cases} \text{GM} - \left|1 - \frac{n^+}{N^-} \cdot P\right| & \text{if } N^- > 0 \\ \text{GM} - P & \text{if } N^- = 0 \end{cases} \tag{4.2}$$

where n^+ stands for the number of minority class instances, and N^- stands for the majority class instances selected in a solution. P is a penalization factor that accounts for the importance of balancing both classes; [28] recommends this parameter to be set to 0.2.

To find the right balance between exploration and exploitation, EUS uses a well-known algorithm called CHC [39] that uses the heterogeneous uniform crossover GA procedure between two chromosomes. For that, exactly half of the different genes are interchanged between the parents. In order to prevent incest, the mating is only considered if the diversity of both parts is sufficient [30].

As mentioned in Sections 2.3.4 and 4.1.1, it is often essential to make sure that the datasets used for classifying are as diverse as possible. A relation between diversity and single-class measures was found in [40], having a positive impact on positive class classification and global performance measures such as the AUC. To promote diversity, EUS prefers solutions that are different from the best solutions in previous generations. As it is a preprocessing algorithm, it is assumed that base classifiers learned from these specially-treated datasets are ultimately more diverse. For that reason, the authors of [30] proposed a modification from the original fitness function (4.2). For this modification to be implemented, the diversity between the solutions must be measured, which is done by introducing the Q-statistic [41], which is applied by comparing two solutions with each other. The result refers to the diversity between two solutions and ranges from -1 to 1, lower values indicating greater diversity. With two binary vectors (solutions or chromosomes shown in 4.1), the Q-statistic can be calculated as follows [30]:

$$Q_{i,j} = \frac{N^{11}N^{00} - N^{01}N^{10}}{N^{11}N^{00} + N^{01}N^{10}} \tag{4.3}$$

where N^{ab} is the number of instances with value a in the first and with value b in the second vector; if a and b are equal, both datasets do or do not include the same data sample. $Q_{ij} = 0$ means that both solutions are statistically independent, whereas $Q_{ij} = \pm 1$ means that both solutions exist of the same data samples and thus are statistically depended. EUS compares the diversity between a candidate chromosome and the previously best chromosomes from each generation. As Q-statistics is a pairwise measure, a global maximum Q of all Q_{ij} is considered to aggregate the pairwise comparisons.

$$Q = \max_{i=1,\ldots,t} Q_{i,j} \tag{4.4}$$

where V_j is the candidate solution and V_i represent all previous solutions from the first to the current one (t). The authors of [30] also introduce a weighting factor β that changes every iteration and the imbalance ratio ($IR = N^-/n^+$) to further increase the diversity (more information in can be found in [2]). The weighting factor and the new fitness function using all diversity evaluation steps can be defined as follows:

$$\beta = \frac{N - t - 1}{N} \tag{4.5}$$

$$\text{fitness}_{\text{EUS}_\text{Q}} = \text{fitness}_{\text{EUS}} \cdot \frac{1.0}{\beta} \cdot \frac{10.0}{IR} - Q \cdot \beta \tag{4.6}$$

The first iteration needs special consideration, as there are no vectors to compare with the actual candidate solution. For this reason, the fitness value for $t = 1$ is calculated by the original Equation (4.2). After that, this fitness function can be used as stated in (4.6).

With all the underlying classification and sampling background now explained, the actual performance evaluation would be possible. However, to check the results for any statistically significant difference, several statistical tests need to be performed to draw reliable results from the calculated performance evaluation. These tests will be the focus of the next section to prepare for the implementation part of this thesis.

4.3. Statistical Testing of Algorithms

In order to compare two classifiers, the typical approach would be to study their average performance measures. However, reliable conclusions can only be drawn if sufficient statistical evidence is found that the two algorithms of interest have or have not achieved different performances in relation to the chosen score. The main goal is to decide whether the differences between algorithms are due to chance or real. The natural question to ask would be: How representative is the empirical difference of two classifiers on a given dataset, using the difference between their mean performances? This question can be answered by using a method called statistical hypothesis testing which is an analysis in which a hypothesis, also referred to as the null hypothesis or H_0, is assumed. Statistical tests are then used to reject or accept this assumption. H_0 states that both algorithms perform equally well based on the selected score and, a priori, is assumed to be true. The alternative hypothesis, or H_1, on the other hand, states that the two algorithms behave differently. The statistical test can result in a correct or incorrect decision (see Table 4.2). A type I error is the probability of deciding that H_0 is rejected when, in fact, H_0 is true. This error is called the level of significance (α) and is frequently set to 0.05. In practical terms, this α means that there is a 95% chance that the resulting statistical difference is real and not due to chance [42]. In order to make a statistical decision, the *p-value* needs to be calculated. This p-value resembles the probability of obtaining the statistic's observed value, assuming that H_0 is true. In other words, a small p-value (smaller than α) means that the observed outcome is possible, but very unlikely under the assumed null

		Decision	
		Do not reject H_0	**Reject H_0**
Reality	**H_0 is true**	Correct	Type I error $\alpha = P(reject\ H_0 \mid H_0\ is\ true)$
	H_0 is false (H_1 is true)	Type II error $\beta = P(do\ not\ reject\ H_0 \mid H_0\ is\ false)$	Correct $Power = P(reject\ H_0 \mid H_0\ is\ false)$

Table 4.2.: Errors in the decision of statistical tests, by [43]

hypothesis. Therefore, it is assumed that H_0 was a wrong initial assumption, which leads to a rejection of the null hypothesis.

However, some assumptions have to be considered when using statistical tests, depending on the test used. Two broad classifications of statistical tests exist, *parametric* and *nonparametric* tests. Parametric statistical tests are used when the distribution of the underlying population from which the sample was taken can be assumed. The most commonly used parametric assumptions are that the data is approximately normally distributed and has homogeneity of variance. As mentioned in Section 3.3, a common validation type in the class imbalance problem is to use k-fold-cross-validation. The result of a k-fold-cross-validation is not completely independent, ultimately leading to a set of classification results that neither presents a normal distribution nor a homogeneity of variance [28]. According to the recommendations made in [28, 44, 45], nonparametric tests are used in this thesis and will be presented in the following parts. Figure 4.8 illustrates an overview of different statistical tests, this is by no means a comprehensive list of all available statistical tests, but it represents the more appropriate tests for the use in classification problems [46].

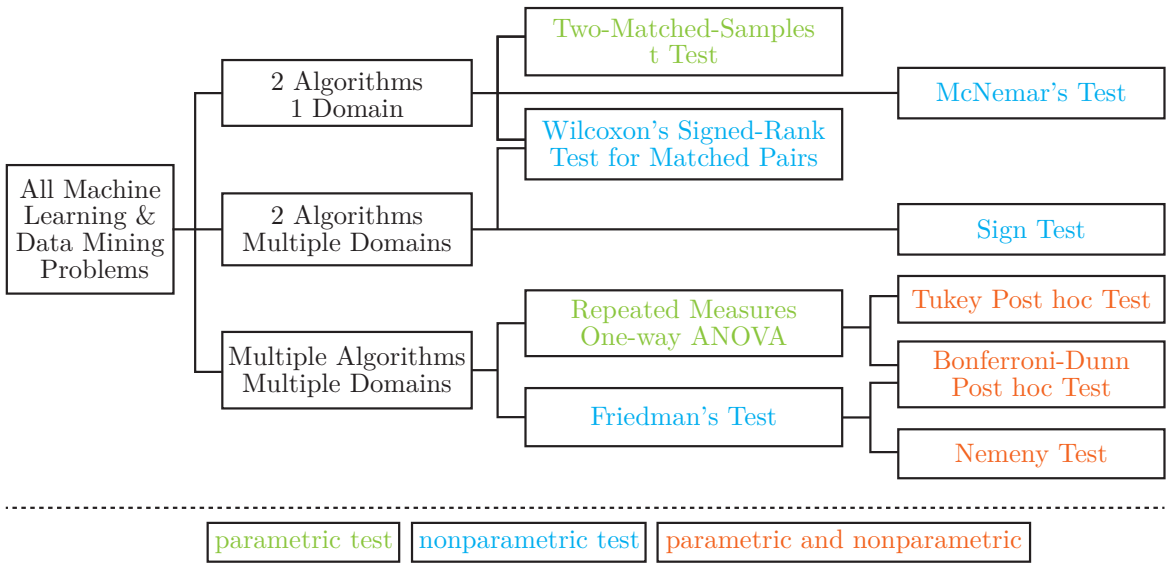


Figure 4.8.: Overview of statistical tests, by [46]

For the nonparametric statistical tests considered in this thesis, there are two different types of comparison: pairwise comparisons and multiple comparisons. As the authors of [2] suggest, multiple comparisons will be executed with the Friedman Test to detect significant differences between a set of algorithms. If differences are found, a post hoc test can determine where those differences are. For instance, the best algorithm could be checked against all the others ($1 \times n$ comparison) if it is significantly different under the Friedman Test. In order to test pairwise comparisons, an altered form of the Wilcoxon Signed-Rank Test will be chosen to determine whether there are significant differences between a pair of algorithms.

4.3.1. Nonparametric Statistical Tests

The Friedman Test is a nonparametric statistical test to compare more than two related samples with each other. The null hypothesis states that all algorithms are equivalent, therefore, a rejection of this hypothesis would mean that at least one of the tested algorithms performance measures is different from the others. This test ranks the algorithms for each dataset separately according to the used performance measure (r_i^j), from the best performing algorithm to the worst in ascending order. After obtaining these ranks, the Friedman Test uses the algorithms' average ranks (R_j) to check against the null hypothesis. As H_0 states that the algorithms are equal, this test assumes that the average ranks should be equal as well. The equations used by the Friedman Test are:

$$\textit{Average ranks} : R_j = \frac{1}{N}\sum\nolimits_i r_i^j \tag{4.7}$$

$$\textit{Friedman Statistics} : \chi_F^2 = \frac{12N}{k(k+1)} \sum_{j=1}^{k} \left(R_j^2 - \frac{k(k+1)^2}{4} \right) \tag{4.8}$$

where N and k are the rows and columns of the input matrix. For the EUS example, these inputs can be explained with this example: N undersamplers each rate k different classifiers. Do any of the k classifiers perform consistently higher or lower than the others?

This Friedman statistic follows a Chi-squared distribution (χ^2) with $k-1$ degrees of freedom for large N and k. Due to this distribution, a smaller N (usually ≤ 10) and k (usually ≤ 5) the χ_F^2 and the resulting p-value starts to get less reliable and should be avoided.

A rejection of H_0, solely states that there is a significant difference between at least two of the tested algorithms. In order to pinpoint the specific differences, so-called post-hoc-tests are needed. The authors of [28] use the Holm's method, a procedure for multiple comparisons for a candidate algorithm. Usually, the best algorithm is chosen as the candidate algorithm compared with all other algorithms in the test. The statistics used

for the comparison is:

$$z = \frac{(R_i - R_j)}{\sqrt{\frac{k(k+1}{6N}}} \tag{4.9}$$

This z value is then used to find a corresponding probability from the table of a normal distribution, which is then compared with an appropriate α. Unlike the more common and easier to perform Benferroni-Dunn test, the α is recalculated for each comparison, leading to a test result that controls the family-wise error rate (FWER) and rejects more hypotheses. The FWER is the probability of making false discoveries when performing multiple hypotheses tests. As a consequence, despite the higher implementation complexity, there is never any reason to use the simpler Bonferroni-Dunn correction, since it is always outperformed by the slightly more elaborate Holm correction [28].

Finally, the Wilcoxon Signed-Ranks Test can be used for pairwise comparison. It aims to detect any significant differences between the two investigated algorithms. It is a pairwise comparison procedure that can be briefly explained in the following steps [43]:

1. Formulate a H_0 and a H_1, e.g., "The median difference of the algorithms in observation is zero." Also, the significance level (usually $\alpha = 0.05$) and the number of measurements (n) for each algorithm should be noted for a later step.
2. The differences of each measurement are calculated (d^i).
3. After that, the absolute values of the differences are created.
4. The absolute values are ranked, starting with the smallest to the largest value (1 to n). In case of ties between two or more d^i's, each value receives a rank equal to the average of the ranks they span.
5. These ranks are then summed up for differences that resulted in a positive value to get T_+. The same is done with the negative values for T_-. The used equations are:

$$T_+ = \sum_{i=1}^{k} I(d^i > 0) \cdot rank(d^i) + \frac{1}{2} \sum_{i=1}^{k} I(d^i = 0) \cdot rank(d^i) \tag{4.10}$$

$$T_- = \sum_{i=1}^{k} I(d^i < 0) \cdot rank(d^i) + \frac{1}{2} \sum_{i=1}^{k} I(d^i = 0) \cdot rank(d^i) \tag{4.11}$$

6. Calculate the W_{stat} by finding the minimum of both values $min(T_+, T_-)$.
7. Finally, W_{stat} is compared with a critical value (W_{crit}) taken from a Wilcoxon Signed-Ranks table. This table provides critical values for different number of measurements and significance levels for W_{crit} up to $N = 25$.

However, the Wilcoxon Signed-Rank Test is only considered for paired samples, meaning that the samples compared need to be related or matched samples. For instance, repeated measures on the same subjects can be used. As the results calculated from different

classifiers are not paired, an altered form of the Wilcoxon Signed-Rank Test can be used. The "Mann-Whitney U Test" (also called "Wilcoxon Rank-Sum Test") is a nonparametric test that considers independent samples for statistical testing and is therefore used in this thesis' implementation part.

The mentioned tests and methods are by no means a complete and comprehensive list of every available statistical test but are sufficient to present the general idea behind statistical testing. All the above mentioned EUS steps and subsequent statistical testing methods from this section will be used in the implementation part of this thesis. The next section presents the dataset being used.

4.4. Dataset

Testing the methods mentioned above for intrusion detection systems is a difficult task. Popular benchmarking datasets in the field of IDS research, such as the KDD-99 [47] and the MIT Lincoln Laboratory's DARPA [48], are becoming obsolete for several reasons. Although they were extremely relevant for the time of publication, they are now too old for modern IDS and have lost their relevance over time. The KDD-99 dataset was created using a Solaris-based system to collect a wide range of data. One problem was that the Solaris Operating System had a minimal market share compared to other operating systems, such as Unix or Windows. The most significant problem is that the technology used in the late 1990s has very little resemblance to the technology used in current times. Changes such as the conversion from 32-bit to 64-bit systems had a far-reaching impact on modern systems' exploitability. At the time of the creation of these pioneering IDS datasets, there was no standardized way of evaluating IDS performances; another concern with various researchers [49]. Various artifacts found in those datasets led to significant variations of classification results with modern algorithms. Since there was no viable alternative, they are still used as the basis for many IDS validations by many researchers.

To overcome this long stretch without modern datasets for IDS research, the Australian Defence Force Academy created several new datasets, one of which this thesis will take a deeper look at, called the Australian Defence Force Academy - Linux Dataset (ADFA-LD) (downloadable at [50]). This dataset was chosen because of a variety of reasons. First, this dataset was generated based on modern computing infrastructure, namely the Ubuntu Linux Server Version 11.04 [51]. It also contains up-to-date attack methodologies and can thus reflect on the latest characteristics and realistic performance of state-of-the-art attacks. This dataset was made specifically for AIDS; accordingly, different attack vectors for the Ubuntu operating system were chosen. Those attack vectors are dependent on various software running on the underlying operating system. For instance, to enable web-based attacks, the webserver Apache Version 2.2.17 [52] running PHP Version 5.3.5 [53] had to be installed and enabled. Other exploits used a web-based collaborative tool called Tiki Wiki [54]. The services FTP, SSH and MySQL Version 14.14 [55] were started as services on their default ports to add several commonly used vulnerabilities. This fully patched local Linux server represented a reasonable generalization of a server, offering

commonly used tasks such as file sharing, database services, remote access, and web server functionality [56].

Different payloads were used to generate malicious traffic in the ADFA-LD dataset to exploit the available known vulnerabilities. These payloads consist of password brute force, adding new superusers, Java, and Linux Based Meterpreter C100 Webshell. The vectors used for each payload are shown in Table 4.3.

Payload/Effect	**Vector**
Password brute force	FTP by Hydra [57]
Password brute force	SSH by Hydra [57]
Add new superuser	Client side poisoned executable
Java Based Meterpreter	Tiki Wiki vulnerability [58]
Linux Meterpreter Payload	Client side poisoned executable
C100 Webshell	PHP Remote File Inclusion vulnerability

Table 4.3.: ADFA Linux Dataset Attack Structure from [59]

The dataset creators have carefully considered contemporary methods used by penetration testers and hackers and have found a delicate compromise between the vulnerability of the target system and the required realism. As remote exploits are a relatively rare find in fully patched systems, only the TikiWiki remote code execution vulnerability was engineered into the dataset. This small flaw represents a realistic vulnerability in an otherwise well-configured server. Datasets with overly porous and trivially exploitable servers are considered to be utterly artificial, thus not desirable in the field of IDS. Meterpreter are payloads generated by the most commonly used open-source hacking toolkit available called Metasploit. It is widely used by penetration testers, security professionals, and hackers; therefore, it is used to create multiple payloads mentioned in Table 4.3, such as adding new superusers, Java and Linux based meterpreter payloads. The C100 shell is part of the webserver and enables the attacker to use an illicit GUI interface through a web browser to manipulate the underlying operating system. This collection of attacks allowed the creators to model a representative dataset for IDS evaluation with current practices and recent attack sets [56].

The ADFA-LD dataset consists of a set of system call traces, recorded with the Unix program `auditd`. System call traces are considered as the most accurate way to detect malicious activities for anomaly-based systems and are widely used in the IDS research community [60]. The activities recorded to create the dataset, ranging from normal web browsing to document preparation. They are separated into normal training data traces with a file size of 300 Bytes to 6 Kilobytes and normal validation data traces that range from 300 Bytes to 10 Kilobytes. This differentiation of file sizes was done to create an effective trade-off between processing time and data fidelity. Since this thesis questions if EUS is a suitable substitution to other undersampling methods, the authors of [20] suggest combining all data traces to get a better-suited dataset. As a result, 5206 normal behavior traces are used and will be denoted as the majority or the negative class. In

order to create the attack data, ten attacks per attack vector shown in Table 4.3 were executed. This resulted in a total of 746 attack data traces denoted as the minority or positive class. Together, these 5952 system traces form a dataset with an imbalanced ratio of 1:6.98 [56]. The dataset is structured in subfolders separating attack data and normal data. Every activity is stored in other subfolders within these folders, containing multiple text files that represent system calls traces. These traces refer to a sequence of single system calls for a privileged process. For every system call, there is a unique ID, called system call identification [61].

This presentation of system calls cannot be used directly for a pattern recognition process. Since the system call traces are different in length, they need to be preprocessed beforehand. The authors of [61] proposed three different ways to bypass this problem: considering the trace lengths, usage of common patterns, and counting the frequencies. However, since trace lengths tend to be an inefficient way to find anomalies and the usage of common patterns is too time consuming, the authors suggest using a frequency based counting method. The implementation of this method will be presented in Section 6.1.

5. Implementing Evolutionary Undersampling

This chapter provides an overview of the steps necessary to implement the EUS algorithm. Starting with the tools and environment used in Section 5.1, the implementation is then explained by using the prepared ADFA-LD dataset in Section 5.2.

5.1. Preliminary: Environment

The eintire process of EUS preprocessing and the subsequent steps explained in Section 2.3.2 are implemented with the Python programming language. Python has emerged over the last couple of years as the first choice for many scientific computing tasks due to the vast amount of packages and the active ecosystem that has emerged around the data science community. Third-party packages such as *NumPy*, *Matplotlib*, and *IPython* are just a few examples of the tools that lead to success in scientific computing [62]. As it is often difficult to keep track of all the used packages and versions of these packages in Python, it is recommended to use virtual environments (venv) in large Python projects. A venv can be explained as a directory that contains a specific collection of packages used for a Python project. This is especially important when different projects need different versions and dependencies of packages. A common solution for this problem is the Anaconda Distribution, which was used as this project's environment manager. For actual code development, a mix of the JetBrains PyCharm integrated development environment (IDE) (mainly for debugging) and JupyterLab was used. JupyterLab, the successor of IPython, is a web-based user interface for live runnable code with narrative text, equations, images, and interactive visualizations. It can be used to write a program iteratively, and test and debug each part of the code one by one. Variables can be stored, visualized, and interactively manipulated. The files are called Jupyter notebook files (or IPython files), and are structured with simple JSON, which contains text, source code, rich media output, and metadata. When debugging more complex functions, PyCharm was the superior software to use in this study. Table 5.1 shows the software used, and the Python libraries alongside their version and literature remarks.

A combination of Python libraries was used in this thesis to make implementing certain parts easier; these are explained in the following paragraph. First, *NumPy* (short for Numerical Python) is one of the most important packages for numerical computing and, thus, data science. It provides an easy-to-use and efficient C API for multidimensional array operations used to store all dataset information, preprocessing, and classifier results. *NumPy* also provides different array algorithms, such as sorting, unique, set, merging, and

Software	Version	Literature
JetBrains PyCharm IDE	2019.3.3	[63]
JupyterLab	1.2.6	[64]
conda	4.8.2	[65]
python	3.8.1	[66]
numpy	1.18.1	[64]
matplotlib	3.2.0	[67]
pandas	1.0.1	[64]
scikit-learn	0.22.2.post1	[68]
imbalanced-learn	0.6.2	[33]
scipy	1.4.1	[69]
statsmodels	0.11.1	[70]
pickleshare	0.7.5	[71]
Python Intrusion Detection	0.1dev	[20]

Table 5.1.: Details on version information of software utilized in this thesis

joining. Next, the *matplotlib* library was used for plotting two-dimensional data visualizations, including under- and oversampling data, ROC plots, and other performance-related plots. *Pandas* provides high-level data structures and functions to simplify working with structured and tabular data in an easy, fast, and expressive way. *Scikit-Learn* has become the go-to library for Python machine learning purposes since the start of its development in 2010. It includes submodules for problems including classification, regression, clustering, dimensionality reduction, model selection, and preprocessing. The *imbalanced-learn* package deals with imbalanced datasets and offers different under-, over- and hybrid sampling methods. *Scypi* provides a collection of packages addressing different scientific problem domains. These problems contain differential equations, linear algebra, interpolation, and various statistical tests. Together, *NumPy* and *SciPy* form a well-suited computational foundation for most of the traditional computing applications. *Statsmodel*, adds a wider variety of statistical tests to the available toolset. Finally, *pickleshare* was used to store results of time-consuming calculations to the local file system. This built-in Python package stores variables efficiently in binary format (also called serialization) [64].

In addition to the different publicly available Python libraries mentioned above, some parts of the "Python Intrusion Detection" repository by the author of [20] were utilized in this implementation. Christian Promper's work helped implement several parts of this work, such as preprocessing the ADFA-LD dataset and comparing different classifiers on this particular dataset. It also provided a solid foundation on which classifiers parameters to choose for classifiers in the evaluation part of this thesis.

Google Cloud Platform (GCP) was chosen as a cloud provider in order to gain more computing power. This service offers many sub-services such as App Engine, Compute

Engine, Kubernets Engine, and different other tools and programs. For this project, the Compute Engine was used. This service allows the creation of Virtual Machines within the GCP infrastructure. GCP offers 300$ for the first project created on the platform, which can then be used with a wide variety of services; the maximum VM that can be created was a ten core processor machine with 8GB RAM. More powerful machines require an upgrade from the free credit version; therefore, this maximum configuration was chosen for the implementation, making it possible to create a JupyterLab instance directly with one click. As the underlying Operating System, Ubuntu 18.04 was chosen.

5.2. Algorithm Implementation

This section will present how EUS is implemented with the prepared Python toolset mentioned before. To do so, it is important do understand the underlying steps of genetic algorithms, as they are an essential component of EUS. As a reminder, GAs are heavily inspired by the Darwinian theory of evolution, where the fittest individual's survival leads to the passing on the fittest genes to the next generation. These algorithms follow the basic steps of every EA, including *initialization*, *evaluation*, *selection*, *crossover* and *mutation*, as shown in Figure 5.1. Several key components need to be adjusted so that the GA can work as an EUS algorithm and are explained thoroughly in the following section of this thesis. This algorithm's main goal is to generate a new dataset that has removed any imbalance between the two class samples. In the following, the minority class samples are denoted as n^+, and the majority class samples are referred to as N^-. The final implementation is included in the code repository on the storage medium attached to this thesis.

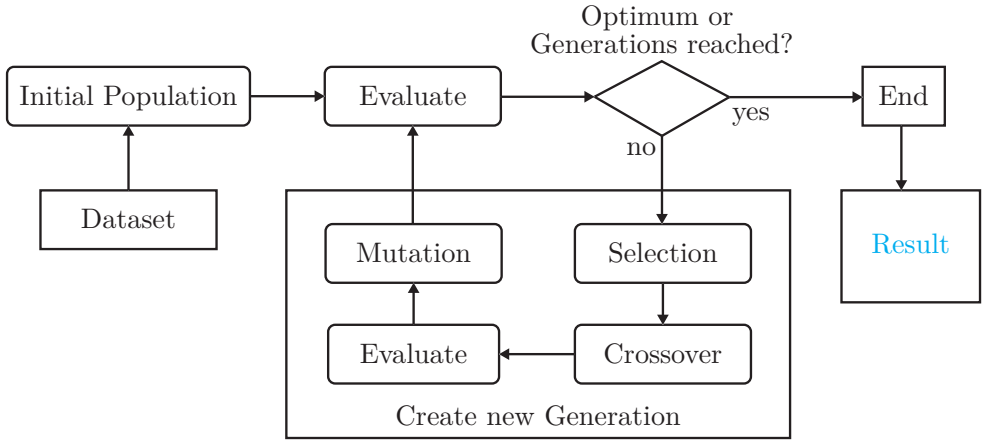


Figure 5.1.: Evolutionary undersampling process

5.2.1. General Steps

As mentioned in Section 4.1.1, every GA starts by creating an *initial population*. In general, a population is a set of multiple solutions called chromosomes. These chromosomes

consist of n genes (one for each instance in the majority class) with two different states: either 0 or 1. A solution in EUS stands for a readily undersampled majority class. In EUS, only the majority class is considered for sampling; therefore, a binary value for each majority class sample is added to the chromosome. If an instance of the majority class is included in the undersampled solution, it is represented by a 1; otherwise, this gene's state is 0. In order to form a perfectly balanced dataset, it is assumed that there should be exactly as many data samples in both of the classes; in other words, the initial population creates a $m \times n$ matrix, where each m represents a solution in the population, and n represents each majority class sample of the used dataset. Figure 5.2 illustrates these different subparts for the ADFA-LD dataset. For his dataset, a chromosome would consist of 5026 genes representing the occurrence of data samples in the undersampled majority class. The index of a gene represents the row of the system call trace in the prepared dataset. The sum of each chromosome results in the amount of the minority class samples, or in case of the ADFA-LD dataset in 746 ($\sum_{i=1}^{n^+} N_i^-$).

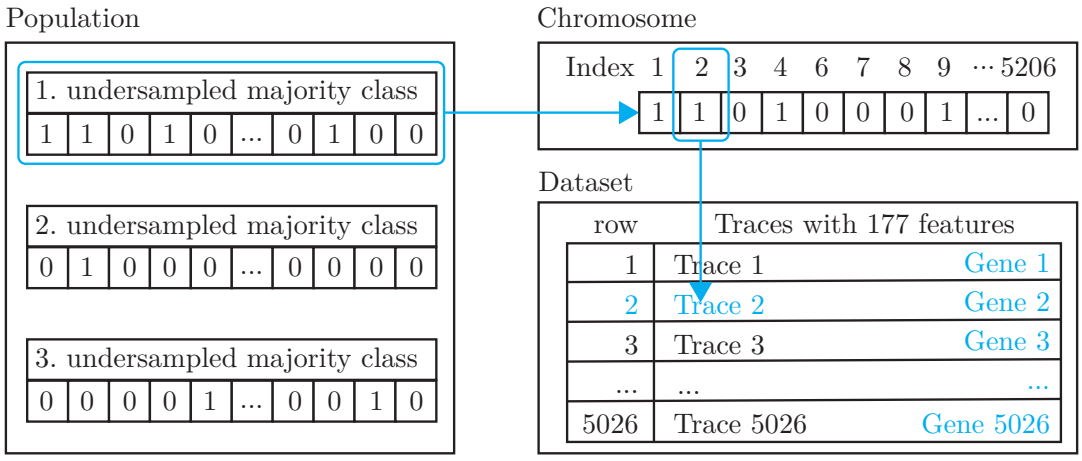


Figure 5.2.: Population, chromosomes, and genes in the implementation

To generate this initial population, the method `initial_population()` was created. This method makes sure that each N^-sample is represented in the solution space of the initial population. A randomly permuted index array was used to assign each chromosome in the population a randomly chosen N^-samples. After assigning every majority sample at least once, the remaining chromosomes were then filled with randomly chosen majority samples.

The next component in an GA is the *evaluation*. This step is one of the most crucial parts of the algorithm, as it is responsible for deciding how "fit" or "good" the solution is, concerning the problem in consideration. In the case of pattern recognition, the objective of finding a suitable solution is generally to maximize or minimize a measurement that helps with the evaluation of classification. A straight-forward method for a fitness function would be to maximize the geometric mean with imbalanced class distributions. As mentioned in Section 3.3, the GM ($\sqrt{\mathrm{TP}_{rate} \cdot \mathrm{TN}_{rate}}$) takes both the accuracy of the majority and minority class into consideration. Therefore, an improvement of the GM would

lead to better classification results in both classes, respectively. This fitness function is the one component that makes EAs highly adaptive to different problems.

One limiting factor of this implementation is that the fitness function needs to be calculated often, illustrated in Figure 5.1. To further improve the fitness function's performance, twice as many chromosomes are generated from each generation as needed and the best-suited chromosomes according to the fitness function will be picked. However, a time-intensive consequence of this method is that these new solutions need to be ranked separately to be able to pick the best-suited ones. After the best chromosomes are picked and mutated, the next evaluation is needed to check if the newly created and mutated solutions have reached the optimum. The number of `fitness_function()` calls is highly dependent on the population size and the number of generations used in the EUS algorithm. The number can be calculated with this equation:

$$fitnessfunction_{calls} = n_{pinitial} + n_g \cdot (n_{pgeneration} + 2n_{pchild}) \tag{5.1}$$

For a chosen population of $n_p = 25$ and generations of $n_g = 500$ this would result in 37525 calculations of the fitness function. As a result of calculating the GM each time, a considerable amount of processing power is needed to run through the fitness function's classifying steps. These steps include combining the chromosome majority samples with the minority samples, splitting dataset into training and testing data, training the classifier, using the trained classifier to predict testing data, and finally calculating the predicted data results. To find a trade-off between processing time and classification performance, the authors of [30] suggested a 1NN classifier, as it is fast and relatively resource friendly. To verify this statement, several test runs were made with different k-nearest-neighbor values. A brief overview of this process is shown in Table 5.2 for 500 generations and population size of 25. It was found that increasing the kNN value leads to increased processing time with no significant changes in the achieved GM. Therefore, the 1NN classifier was chosen for further implementation of the algorithm.

	k = 1	**k = 2**	**k = 3**	**k = 4**	**k = 5**
Average GM reached	0,9786	0,9812	0,9831	0,9678	0,9732
Processing time in sec	3308,19	3640,43	3876,58	3914,07	4129,38
Average time \ generation in sec	6,62	7,28	7,75	7,83	8,26

Table 5.2.: Different kNN values used in the fitness function

By using the fitness function introduced in Section 4.2, a maximum value can be reached. Except for the GM, every variable of the equation (4.2) is a constant. Therefore, if no class samples are misclassified and thus the GM resulting in 1 (when TPR and TNR are both 1, so $\sqrt{1 \cdot 1}$), the maximum fitness value is reached. There is no reason to continue generating new chromosomes at this point because other solutions cannot improve this maximum. There may be other solutions that also result in the maximum; however, to save processing time, the algorithm stops generating new solutions when the maximum

value is achieved. Another form to improve the processing time is to include a forced stop of the algorithm after a certain amount of generations were not able to improve the fitness value. For the implementation, the forced stop of the algorithm is chosen as follows: $stop = generations \cdot (\frac{1}{4})$.

The next step in EUS is the *selection* of potential chromosomes for creation of the next generation. This involves the roulette wheel selection method mentioned in Section 4.1.1. To decide which chromosomes are chosen for reproduction, this method assigns each chromosome a probability to be chosen by the roulette wheel proportional to their fitness value. The better the fitness value, the higher the chance to pass the genes to the next generation. In the implementation, *elitism* was used to ensure that the best performing chromosomes were automatically picked for the creation of the next generation. To keep the stochastic and explorative approach of EUS, it is essential to use a small number of elite individuals. The crossover with lesser suited chromosomes was used to create a wider variety of solutions and therefore, positively effecting the explorative side of this algorithm.

Once the different chromosomes are chosen by the `selection()` method, the *crossover* component begins its work. In order to avoid overfitting, undersampling methods try to prevent duplicated data points in the dataset. For the implementation, an altered uniform crossover technique was chosen as illustrated in Figure 5.3.

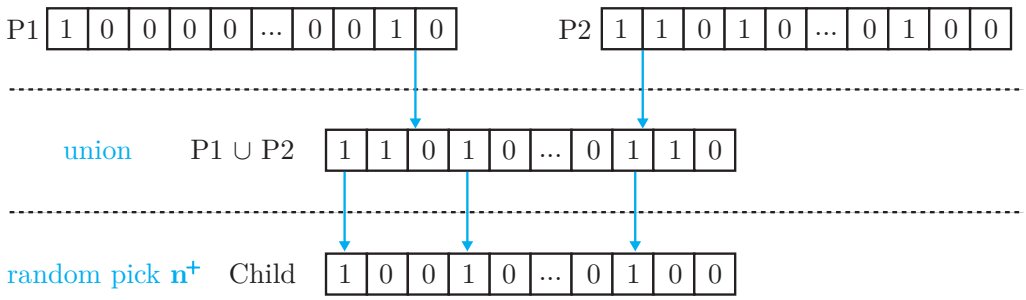


Figure 5.3.: Crossover of two chromosomes

In this method, both parents are first combined (P1 ∪ P2), and then the number of n^+samples is randomly selected from this union to create a child. This process is repeated until twice the population size has been created. After that, the new population is ranked, and only the population size is picked for the next steps, thus rejecting the worst fitness values from the newly created generation. This step was included in the implementation to optimize the GM. The algorithm starts with the mating of the elite chromosomes, then the rest of the selected elements are crossed.

The final step when creating a new generation is *mutation*. This is a way to introduce more variation in the undersampled population by randomly swapping genes within chromosomes. The function `mutate()` has a fixed chance that it will be applied to the chromosome. For this implementation, a mutation rate of 5% was chosen, as it has shown the best results in providing a good trade-off on diversity and randomization. Section 4.1.1 explained that the mutation rate is set to a relatively low threshold in order to

prevent too much randomization. Figure 5.4 illustrates the mutation component.

Figure 5.4.: Chromosome mutation in EUS

Finally, the new generation is created and can be re-evaluated. The best fitness value alongside with its chromosome is stored for the entire runtime of the EUS process. Subsequently, it is checked if the maximum fitness value or the maximum number of generations is reached. The best chromosome is returned and the resampled dataset is then used for the pattern recognition process' next steps.

5.2.2. Promotion of Diversity in the Fitness Function

During the process of implementing EUS, two different versions were created: one without considering diversity between chromosomes (denoted as EUS_{norm}), and the other one with adjustments to the fitness function to promote diversity (denoted as EUS_{div}). When considering diversity, additional steps need to be included in the fitness function. The general idea was already mentioned in Section 4.2. As a reminder in [30], the authors stated a relation between diversity and single-class performance measures. Diversity has a positive impact on the minority class classification and global performance measures such as the AUC. Since EUS is a preprocessing algorithm, it cannot directly impact the diversity of the classification output. For this reason, it is assumed that diversity among different chromosomes is preferable. Base classifiers learned from these chromosomes are therefore considered to be more diverse. To simplify explanation in subsequent parts, again the adapted Equation (4.6) is shown: $\text{fitness}_{\text{EUS}_\text{Q}} = \text{fitness}_{\text{EUS}} \cdot \frac{1.0}{\beta} \cdot \frac{10.0}{IR} - Q \cdot \beta$

To provide this diversity in undersampled solutions, the global Q-Statistic (4.4) over all generations is considered. The Q-statistic is a pairwise comparison (4.3) that compares the diversity of the candidate chromosome with all previous solutions ($\max_{i=1,\dots,t} Q_{i,j}$) and results in a value between -1 and 1. As shown in Equation (4.6), this Q is subtracted from the original fitness value $\text{fitness}_{\text{EUS}}$. Thus, it is handled as a penalty, meaning if $Q = 0$, it is not penalized because it is statistically independent of previous solutions and, therefore, a diverse solution. The weighting factor β (4.5) changes with every iteration, penalizing later generations less than earlier ones.

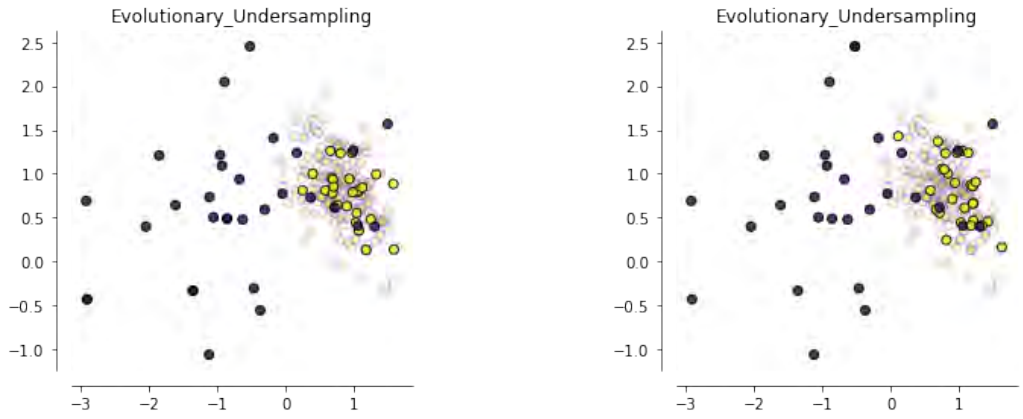
With a higher number of generations, this method's consequence is increased processing time because the calculation of all previous solutions quickly compounds. Despite the slower processing time, tests have shown that resulting performance measures of the EUS_{div} algorithm are superior compared to the EUS_{norm}; therefore, this method was used for the following parts of this thesis. The differences between the two methods are illustrated in Table 5.3.

As a result of the preprocessing steps in EUS, an undersampled dataset is created that promotes diversity throughout generations, and the fittest individual was chosen as the

	AUC	**ACC**	**F1**	**GM**
EUS_{div}	0,9337	0,9198	0,7486	0,9335
EUS_{norm}	0,9320	0,9130	0,7339	0,9317
Difference	0,0017	0,0069	0,0147	0,0018

Table 5.3.: Comparison of two different EUS implementations

best possible solution for the particular problem. To illustrate the results, two plots similar to Section 3.2 were created in Figure 5.5. Figures 5.5a and 5.5b represent the same dataset with different results from EUS. As these solutions are based on a stochastic process, the solutions vary between multiple runs of the algorithm. Therefore, it is recommended to use a k-fold-cross-validation mechanism to get a robust result. This process should be repeated multiple times and the average performance should be reported.



(a) (b)

Figure 5.5.: Different undersampling results of EUS, side by side

6. Evaluation

This chapter will deal with the evaluation of the implemented EUS algorithm in imbalanced datasets. As this algorithm is just a small part of a classifier's performance evaluation, several additional methods need to be taken into consideration to obtain the desired result. Figure 6.1 illustrates the entire process to gather performance results that can subsequently be used for statistical testing. The next section will explain those methods and classifiers that are going to be used. Performance metrics are an essential factor to consider in evaluation results, as not all of them are suitable for unbalanced datasets. Therefore, the performance metrics used are explained in this section. To illustrate the impact of sampling methods, the ADFA-LD dataset will first be classified without any undersampling applied. Then different undersampling methods will be used to show the positive effects of preprocessing. Finally, this chapter will apply various statistical tests to the evaluation results to see how EUS performs compared to traditional undersampling methods.

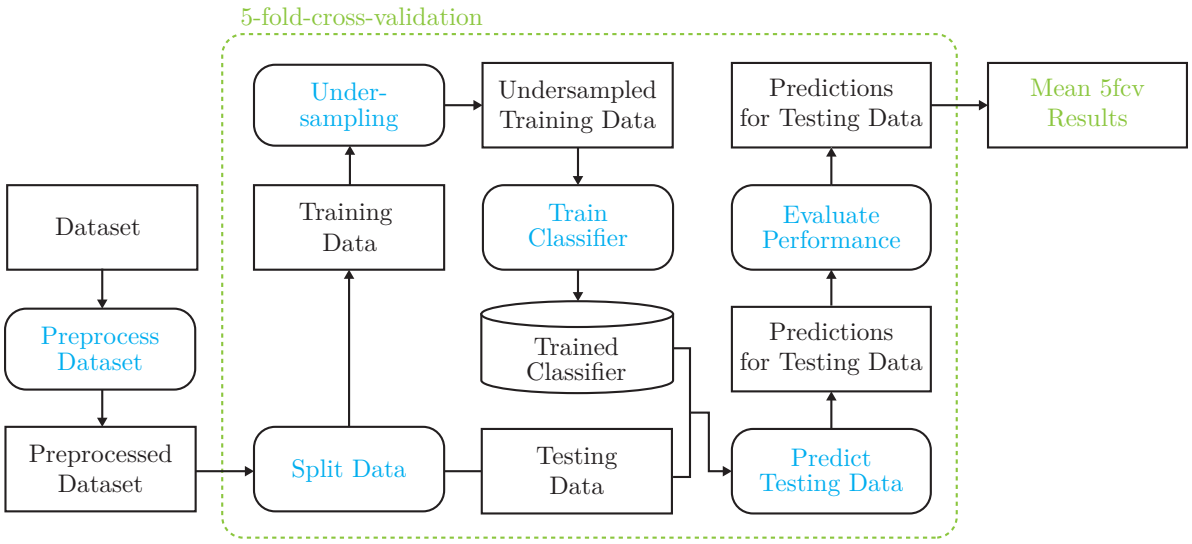


Figure 6.1.: Big picture of the undersampling and classification process

6.1. Used Methods, Classifiers and Performance Metrics

The following section will present the underlying subcategories that are applied in the steps *preprocessing data*, *splitting data*, *undersampling data*, *training classifiers*, *predicting test data*, and *evaluation of performance*, all shown in Figure 6.1. The general idea when evaluating a classifier can be divided into some basic actions. First, to obtain a more

robust result, it is always recommended to run the executed test multiple times and average results to eliminate certain arbitrariness, and eliminate any possible outliers. Second, as mentioned above, to avoid any bias that can arise when splitting data, cross-fold validation should be used. When dealing with imbalanced class distributions, the training data should then be sampled for every fold. This sampled data will be used to train the chosen classifier and subsequently predict classes in the testing data. The achieved results are then recorded and, at the end of the k-fold-cross-validation, averaged to get the unbiased performance measure. With the iterations of this process mentioned above, a robust result can be achieved and the best overall model chosen. If there are no obvious models that perform better than others, statistical tests will then be used to find significant differences.

Preprocessing and Evaluation Methods

The first step in the pattern recognition cycle is the preprocessing of the available data. To gain a useable data representation of the ADFA-LD dataset, the authors of [20] used a frequency-based counting method for the system trace calls, in which the goal is to find a common sample length. Therefore, the adapted dataset will consist of the same number of features as there are system call IDs. The highest appearing system call ID in ADFA-LD is 340; therefore, 341 features per data sample are created. Initially, for each system call trace, a row with 341 zero values is created. Then, each occurrence of a system call ID found in the trace files increases the feature value by 1. After processing a complete trace, the row is added to a matrix containing normal data or attack data. Ending this process, two $n \times m$ matrices are created with m as the number of data instances and n as the number of features. The authors of [20] also found that certain system call IDs never occurred within all system call traces. Consequently, the generated matrices included redundant information. After removing redundant feature IDs with the column sums equal 0, the number of features could be reduced to 177. This adjustment had no negative effects on the detection rate, but the reduced feature space resulted in higher performance. The final extracted data is illustrated in Figure 6.2 for both normal and attack data. The implementation of this data extraction was used from the above mentioned "Python Intrusion Detection" Python repository by [20].

data sample \ feature	1	2	3	4	5	...	174	175	176	177
1	0	0	0	125	0	...	25	0	0	0
...	530	0	0	6	0	...	48	28	0	126
5206	175	0	5	0	0	...	276	12	0	0

Figure 6.2.: Extracted data samples from the ADFA-LD

In the implementation, this extracted normal and attack matrices are concatenated to create the features variable used for all classification procedures. Alongside, a target variable is created that represents the ground truth necessary for supervised learning,

where each row represents the class of the data sample in the feature's variable (0 for $N^-$and 1 for n^+). As mentioned in Section 4.4, the final dataset consists of 5952 data samples.

Another preprocessing step in pattern recognition is scaling the data to remove possible noise and increase performance by decreasing the variability between features. Therefore, three scaling methods mentioned in Section 3.3 are utilized in the implementation. Tests showed no significant differences between the scaling methods, and this result also corresponds to the test results of those found in [20]. Overall, minor improvements in performance values can be achieved with the arctan scaling method; therefore, arctan scaling will be used for all the following evaluation steps in this thesis. Contrary to interval and z-score scaling, arctan scaling is not directly implemented into the sklearn Python library. To use this scaling method, the dataset is first standardized with the `sklearn.preprocessing.StandardScaler()` method and then scaled by the arctangent equation, $a = 2/\pi \cdot \tan^{-1}(data_{standardized})$.

As it is important to remove any possible bias from split data, a 5-fold-cross-validation is implemented in the evaluation process. Five folds were chosen based on discussions with supervisors as well as a trade-off between processing time and performance robustness. After the data preprocessing steps, the data is split into five folds of each training and test data. This splitting is done with stratified sampling, meaning that 70% training data and 30% test data is created with each split, while the original class distribution in both test sets is maintained. This is illustrated in Figure 6.3. When using the ADFA-LD dataset, this results in approximately 1190 data samples in each split.

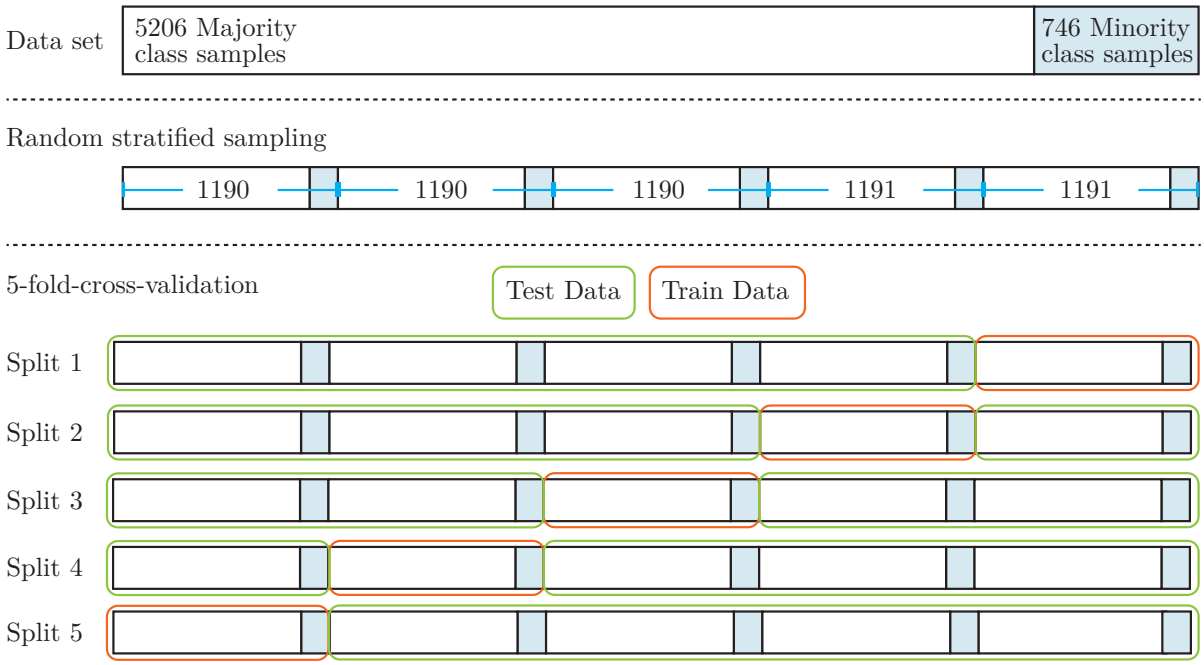


Figure 6.3.: 5-fold cross-validation process after data splitting

The built-in function "StratifiedKFold" from *Scikit-Learn* in the module "model_selection"

was used to execute this task. After processing the five-folds, this process is repeated ten times, and the average value of the performance measures is then taken. The data sample is shuffled to ensure that there are different splits. For further evaluation purposes, each round's results are stored in a separate variable alongside the average performance measures. These individual results for each iteration are used in the statistical evaluation in Section 6.2.

Undersamplers and Classifiers

In order to evaluate and compare the performance of EUS, several state-of-the-art undersampling methods are used to preprocess the split training data before classification. The methods used are implemented with the help of the *ImbalancedLearn* Python package. This package focuses on imbalanced dataset and provides an implementation for common undersampling algorithms and is widely used in scientific work related to this topic. Therefore, the provided methods AllK-Nearest-Neighbor (AllKNN), EditedNearestNeighbor (ENN), NeighborhoodCleaningRule (NCR), OneSidedSelection (OSS), RandomUnderampler (RUS), RepeatedEditedNearestNeighbor (RENN) and TomekLinks (TL) are used with their standard parameters.

Several common single classifiers and ensemble classifiers were chosen based on previous research. For every classifier, some parameters are used to alter the classification boundary called hyper-parameters, which can be chosen freely to tune a model and to improve its classification. To find the best parameters, a grid search was used, which results in the most accurate predictions. This grid search was carried out in [20]; the results of this work were considered and individual classifiers with their parameters were selected and are presented in Table 6.1.

Type	**Classifier**	**Parameters**
Single	DT	default
	MLP	default and solver="lbfgs", hidden_layer_sizes=500
	k-NN	default and n_neighbors=3, weights="distance"
	SVM	default linear and default rbf kernel
Ensemble	Bagging	base_estimator=DT(criterion="entropy") n_estimators=80
	AdaBoost	base_estimator=DT(criterion="entropy") n_estimators=90
	RandomForest	criterion="entropy", max_features=0.4, n_estimators=70

Table 6.1.: Used classifiers with their parameter, from [20]

Some classifiers are used multiple times with different parameters, to generate more variation between classifiers. These classifiers are, Multi Layer Perceptrons (MLP), kNN and SVM. More information about the parameters for each classifier and the definitions of those can be found in [72].

Performance Metrics

There exist different performance metrics, and some are better suited to compare particular datasets than others. In this thesis, metrics suitable for imbalanced data distributions are selected. As mentioned in Section 3.3, it is not recommended to compare classification results with standard metrics, such as true positive (TP), true negative (TN), false positive (FP) and false negative (FN). These metrics lead to misleading results when used with skewed datasets. Therefore, the AUC, ACC, GM, F1-Measure (denoted as F1), FPR and FNR were used. The FPR and FNR were chosen because they allow a better comparison of each detected class, whereas the other measures combine and therefore conceal the class-dependent detection rate results.

6.2. Evaluation on a Real-world Dataset

The previously mentioned sampling methods are used to create results to evaluate. First, it is shown if undersampling can improve the traditional classification process. This is done by presenting the results of those classifiers without any undersampling applied first. The results gained by the evaluation will then be further investigated by statistical tests to show how EUS is performing, compared to other state-of-the-art sampling methods.

6.2.1. Performance Evaluation

All results shown in this process are applied with the 5-fold-cross-validation (5fcv) and 20 iterations to gain a robust result. The first table presents the classification results of chosen classifiers with no undersampling executed during the steps shown in 6.1. The classifiers are denoted with an additional "_d" when the default parameters are used and without "_d" when the grid search parameters are used.

One interesting fact that can be seen in Table 6.2 is that the FNR is relatively high, meaning that a minority class sample was misclassified in 15 out of 100 times on average. Consequently, in an IDS system, this could mean that there is a high chance of malicious network traffic that was classified as normal traffic, which is an undesirably high amount. In the ADFA-LD dataset, this translates to 112 attack traces classified as normal traces. Otherwise, the results are mostly as expected as those classifiers are already optimized for the dataset by the above mentioned grid search. The relatively high accuracy shows again that misclassified classes are not correctly reflected in the accuracy performance measure. Since the primary goal of IDS is to classify all minority class samples correctly, this result is not desirable and needs to be addressed.

Classifier	AUC	ACC	F1	GM	FNR	FPR
DT_d	0,8970	0,9541	0,8175	0,8938	0,1790	0,0269
k-NN	0,9282	0,9684	0,8741	0,9267	0,1254	0,0181
k-NN_d	0,9196	0,9642	0,8576	0,9177	0,1399	0,0209
MLP	0,9254	0,9682	0,8727	0,9237	0,1317	0,0174
MLP_d	0,9265	0,9693	0,8765	0,9248	0,1306	0,0164
linSVM	0,8964	0,9537	0,8161	0,8931	0,1800	0,0272
rbfSVM	0,9000	0,9623	0,8446	0,8962	0,1831	0,0168
DTBagg	0,9207	0,9712	0,8814	0,9182	0,1469	0,0118
DTBoost	0,9119	0,9676	0,8663	0,9089	0,1625	0,0138
RForest	0,9230	0,9733	0,8893	0,9206	0,1440	0,0099
Average	0,9149	0,9652	0,8596	0,9124	0,1523	0,0179

Table 6.2.: Performance metrics of used classifiers without undersampling

To compare those non-undersampled results with the results of different undersampling methods, the classification is calculated once again, with the same parameters and iterations. Presenting ten classifiers for the eight undersampling methods would result in an overly complex table, therefore Table 6.3 only shows two undersampling methods (the full table is attached in the appendix). Presented are EUS, as it is the main focus of this thesis, and NCR because the it performed well and it offers a good balance between the different classifiers.

Positive development can be observed for the before criticized FNR. For the EUS algorithm, there are only 5% misclassified minority samples, which is a significant improvement compared to the average 15% with the unsampled ADFA-LD dataset. However, this was achieved at the expense of decreasing the accuracy by almost 7%. On the other hand, performance values such as the AUC, GM were improved by using this algorithm. As a result of decreasing the FNR, the FPR was raised, as these measures are typically related. When observing the results of the NCR, an overall improvement of the desired performance measures can be found. Even though the FNR is still relatively high, a reduction of 5% compared to the non-sampled dataset is a promising development. In order to get an general overview of the rest of the undersampling algorithms, the averages of all 10 classifiers are shown per undersampler in Table 6.4.

It is shown that the results are relatively similar with each undersampling method. However, major differences can be observed between the ACC, FNR, and FPR on both RUS and EUS compared to the other sampling methods. This result can most likely be due to using too much randomization in the mutation and crossover steps in the EUS, leading to a result close to the RUS. This problem was identified at a later stage of the project. Since the computing power and the time to complete this work was limited, it was necessary to continue working with the calculated results. Therefore, it was noted and needs to be addressed in future work. However, this result does not interfere with the rest of the

Sampler	Classifier	AUC	ACC	F1	GM	FNR	FPR
EUS	DTBagg	0,9294	0,9146	0,7359	0,9292	0,0509	0,0903
	DTBoost	0,9127	0,9034	0,7060	0,9126	0,0749	0,0997
	DT_d	0,9124	0,9042	0,7073	0,9124	0,0765	0,0986
	MLP	0,9287	0,9216	0,7499	0,9287	0,0617	0,0809
	MLP_d	0,9310	0,9143	0,7361	0,9307	0,0468	0,0912
	RForest	0,9337	0,9198	0,7486	0,9335	0,0478	0,0848
	k-NN	0,9238	0,8976	0,7013	0,9232	0,0412	0,1112
	k-NN_d	0,9102	0,8780	0,6620	0,9092	0,0467	0,1328
	linSVM	0,9246	0,9005	0,7067	0,9240	0,0433	0,1076
	rbfSVM	0,9259	0,9008	0,7081	0,9253	0,0405	0,1076
	Average	0,9233	0,9055	0,7162	0,9229	0,0532	0,1006
NCR	DTBagg	0,9343	0,9644	0,8629	0,9334	0,1058	0,0256
	DTBoost	0,9209	0,9528	0,8236	0,9199	0,1217	0,0365
	DT_d	0,9119	0,9455	0,7995	0,9108	0,1330	0,0433
	MLP	0,9407	0,9573	0,8436	0,9404	0,0816	0,0371
	MLP_d	0,9405	0,9587	0,8477	0,9402	0,0838	0,0352
	RForest	0,9367	0,9665	0,8705	0,9359	0,1030	0,0235
	k-NN	0,9419	0,9533	0,8327	0,9418	0,0733	0,0429
	k-NN_d	0,9317	0,9524	0,8264	0,9313	0,0958	0,0407
	linSVM	0,9206	0,9498	0,8149	0,9198	0,1184	0,0404
	rbfSVM	0,9215	0,9604	0,8464	0,9201	0,1304	0,0266
	Average	0,9302	0,9560	0,8368	0,9293	0,1047	0,0351

Table 6.3.: Performance metrics of used classifiers with EUS and NCR

Undersampler	AUC	ACC	F1	GM	FNR	FPR
AllKNN	0.9304	0.9480	0.8138	0.9302	0.0928	0.0463
ENN	0.9300	0.9531	0.8280	0.9293	0.1013	0.0392
EUS	0.9233	0.9055	0.7162	0.9229	0.0532	0.1006
NCR	0.9302	0.9560	0.8368	0.9293	0.1047	0.0351
OSS	0.9189	0.9637	0.8561	0.9167	0.1414	0.0212
RUS	0.9247	0.9077	0.7213	0.9245	0.0525	0.0980
RENN	0.9308	0.9436	0.8028	0.9307	0.0862	0.0521
TL	0.9188	0.9644	0.8578	0.9165	0.1422	0.0206
Overall avg	0.9259	0.9428	0.8041	0.9250	0.0968	0.0516

Table 6.4.: The average performance of undersampling methods

evaluation, therefore the next steps are executed as planned.

In order to illustrate the differences between non-sampled and undersampled dataset performance measures, the average of EUS is compared with the non-sampled performance measures in Table 6.5.

	AUC	**ACC**	**F1**	**GM**	**FNR**	**FPR**
ADFA-LD no resampling	0,9149	0,9652	0,8596	0,9124	0,1523	0,0179
ADFA-LD with EUS	0,9235	0,9157	0,7402	0,9228	0,0661	0,0869
Difference	0,0086	-0,0495	-0,1194	0,0105	-0,0862	0,0690

Table 6.5.: Comparison of unsampled and EUS performance measures

As shown in the table, the average AUC and GM values increased by almost 1% with EUS. This result was expected because these performance measures are calculated using undersampling methods. The sole purpose of these sampling methods is to reduce the imbalance ratio and, therefore, to improve classification results. Another measure that changed drastically is FPR and FNR. As mentioned before, it is desired to lower the FNR in order to classify the minority class correctly.

6.2.2. Statistical Testing

Finally, the gathered results in the previous chapters are now processed by different statistical tests. These steps are a necessity to prove the correctness of claims made in scientific research. The previous section showed that EUS achieved reasonably close results compared to other undersampling methods, and Table 6.4 presented the average results of all undersamplers used in this thesis and showed that the difference to other undersampling methods is fairly marginal. To ensure that this assumption is correct and does not happen by chance, statistical tests are performed. As mentioned in Section 4.3, scientific work, such as [2, 28, 30, 44] suggest using non-parametric tests for statistical comparison of classifiers. Consequently, the Friedman Test, Wilcoxon Rank-Sum Test, and the Holm method are used for the following parts. The general process of statistical testing is shown in Figure 6.4. All statistical tests used for the implementation are provided either through the *scipy* or *statsmodels.stats.multitest.multipletests* python module.

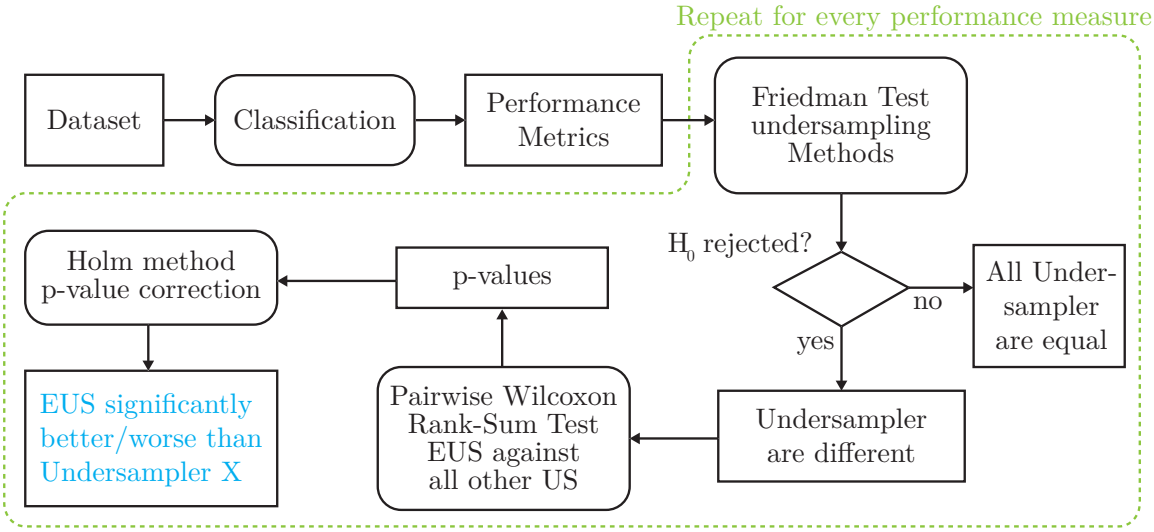


Figure 6.4.: Statistical evaluation process after classification

The resulting performance measures from previous classification steps are now used to execute the Friedman Test. This test is a non-parametric test to compare more than two results with each other. The null hypothesis of this test states that all algorithms are performing equivalent. In order to compare the different undersampling methods, multiple Friedman Tests need to be executed. For each performance metric, every undersampling method is compared with each classifier. The inputs for this test are illustrated in Figure 6.5.

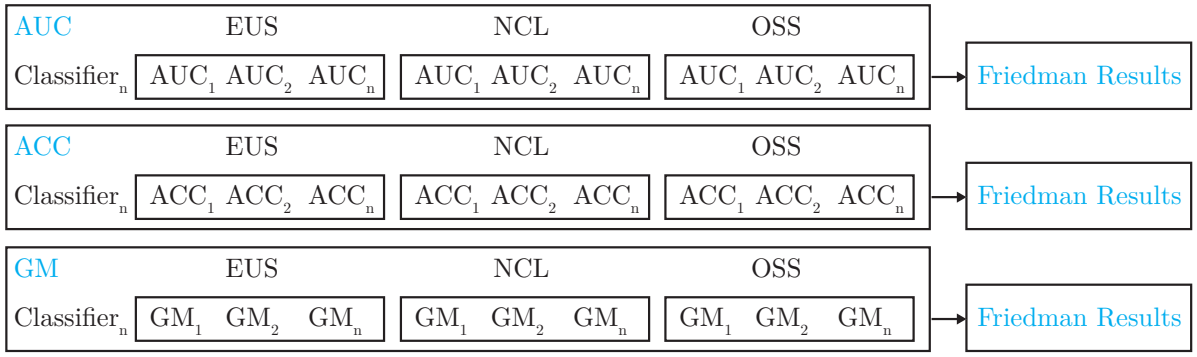


Figure 6.5.: Friedman Test input variables

For every performance measure, a Python dictionary is created containing every classifier's iteration result to compare each undersampling method. In this evaluation, 20 iterations of the performance classification process are used as the classification inputs. Each classifier in every undersampling method is compared to each other, and the results are stored. As an example of the performance measure AUC, the results of the Friedman Test are presented in Table 6.6.

Classifier	Friedman score	Friedman p-value	H_0 rejected
DTBagg	81.6500	6.3447e-15	True
DTBoost	40.0500	1.2314e-06	True
DT_d	87.3000	4.4303e-16	True
MLP	92.1667	4.4384e-17	True
MLP_d	107.9167	2.4830e-20	True
RForest	84.3833	1.7530e-15	True
k-NN	110.2833	8.0200e-21	True
k-NN_d	118.0167	1.9824e-22	True
linSVM	107.4833	3.0534e-20	True
rbfSVM	111.5833	4.3090e-21	True

Table 6.6.: Friedman Test results for AUC

These test results show that all undersamplers for each classifier result in a p-value lower than the level of significance of $\alpha = 0.05$. Therefore, H_0 is rejected for each classifier for the performance measure AUC.

As explained in Section 4.3, when the Friedman Test is rejected, the only knowledge gained is that the tested undersampling methods show a significant difference between at least two sampling methods. To pinpoint the sampling methods that are different, further investigation is necessary. The Python plugin *statsmodels.stats.multitest.multipletests* provides an implementation of the Holm method. The input needed for this method are the p-values of comparison results between undersamplers. To calculate the comparison results for every performance measure, the Wilcoxon Rank-Sum Test was used.

The aim of this thesis is to show if EUS is comparable to other undersampling methods; therefore, EUS will be used as the candidate algorithm for comparison. The input values are illustrated in Figure 6.6. As an example, the Wilcoxon Rank-Sum Test results for the AUC are shown in Table 6.7.

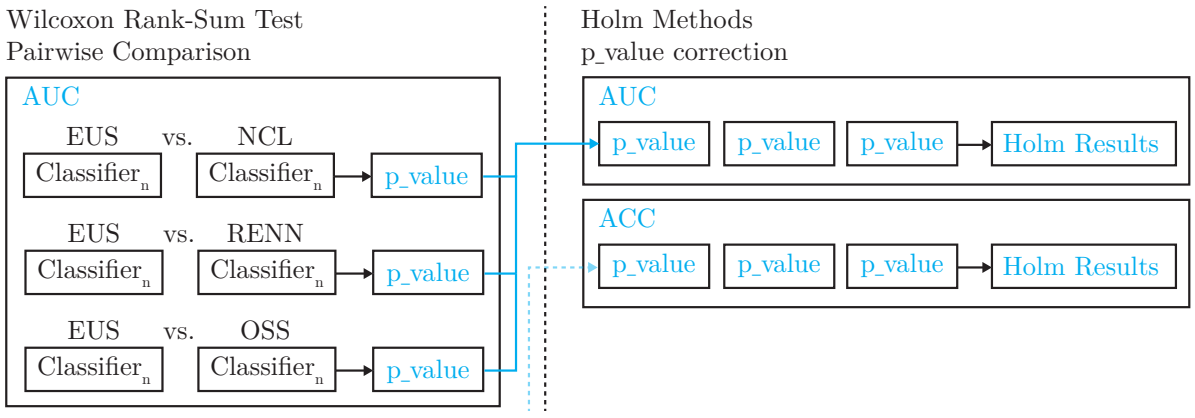


Figure 6.6.: Input values for the Wilxocon Rank-Sum Test and the Holm method.

Comparison	Wilcoxon Rank	Wilcoxon p-value	H_0 rejected
EUS - AllKNN	27	0.0442	True
EUS - ENN	30	0.0699	False
EUS - NCR	31	0.0807	False
EUS - OSS	42.5	0.2979	False
EUS - RUS	43	0.3108	False
EUS - RENN	24	0.0267	True
EUS - TL	42.5	0.2980	False

Table 6.7.: Wilcoxon Rank-Sum Test for the AUC

This result reveals that the AUC shows no significant statistical difference besides AllKNN and RENN. However, the more hypotheses checked, the higher the probability of obtaining Type I errors or false positives, which is not desirable. Generally, the solution to this problem is to use p-value correction controls for either the FWER or false discorvery rate (FDR). This can be done with the Holm method, and its results are presented in Table 6.8.

Compare	p-value corrected	avg difference	H_0 rejected
EUS - AllKNN	0.2651	-0.0071	False
EUS - ENN	0.3493	-0.0067	False
EUS - NCR	0.3493	-0.0069	False
EUS - OSS	0.8938	0.0044	False
EUS - RUS	0.8938	-0.0014	False
EUS - RENN	0.1871	-0.0075	False
EUS - TL	0.8938	0.0045	False

Table 6.8.: Corrected p-values with Holm method for AUC

The Holm method shows the corrected p-values, which changed the results for AllKNN and RENN. Additionally, the average difference in performance measurements between EUS and the other undersampling method is shown. The results for all undersampling methods are presented in Table 6.9. As a reminder, H_0 states: there is *no* significant difference between both undersampling methods. Whereas, H_1 states that: there *is* a significant difference between both undersampling methods. An "a" in this Table denotes that H_0 is accepted, and a "r" refers to a rejection of the null hypothesis, meaning that there are significant statistical differences. As EUS is the algorithm to compare, a rejection with a + symbol implies that the EUS model performs better than the algorithms in comparison. In contrast, a rejection with a - symbol implies that the EUS model performed worse than the method compared.

Comparison	AUC	ACC	F1	GM	FNR	FPR
EUS - AllKNN	a	- r (-0.043)	- r (-0.098)	a	+ r (-0.040)	- r (0.054)
EUS - ENN	a	- r (-0.048)	- r (-0.112)	a	+ r (-0.048)	- r (0.061)
EUS - NCR	a	- r (-0.050)	- r (-0.121)	a	+ r (-0.051)	- r (0.066)
EUS - OSS	a	- r (-0.058)	- r (-0.140)	a	+ r (-0.088)	- r (0.079)
EUS - RUS	a	a	a	a	a	a
EUS - RENN	a	- r (-0.038)	- r (-0.087)	a	+ r (-0.033)	- r (0.048)
EUS - TL	a	- r (-0.059)	- r (-0.142)	a	+ r (-0.089)	- r (0.080)

Table 6.9.: Holm method overview of all performance measures

This final Holm method result indicates that evolutionary undersampling is comparable to other undersampling methods when the AUC and the GM are investigated. EUS seems to provide worse results in ACC, F1-Measure and FPR. As a result of the worse FPR, the FNR is better compared to almost all undersamplers except RUS. Indeed, between EUS and RUS, there is no significant statistical difference, which undermines the statement mentioned above that there is too much randomization within the implementation of EUS. The promising performance in GM results points towards the optimizing of the GM in EUS. As the algorithm is trying to improve the GM in its fitness function.

7. Conclusion and Outlook

This thesis summarizes the work and research conducted on evolutionary undersampling (EUS) during the five-month stay at the Bowling Green State University, Ohio. In particular, the performance of EUS and other undersampling methods has been compared and evaluated in the field of intrusion detection.

The main problem in a modern computer network is that the encountered data distribution of malicious and benign traffic is extremely unbalanced, significantly increasing the difficulty of classification. Based on a real-world intrusion detection dataset, it was shown that the EUS algorithm used during the pattern recognition process is capable of increasing the detection performance of malicious traffic in an IDS. This was demonstrated by using various traditional supervised classification algorithms with the relatively novel and highly imbalanced ADFA-LD dataset. This imbalance was reflected in the test results and addressed by applying preprocessing steps on the dataset before classification. The underlying techniques and methods for both resampling and classification of these imbalanced datasets were presented thoroughly in a state-of-the-art literature review. While the skewed distribution of the dataset was removed or eliminated, these methods were able to keep the features necessary for malicious traffic classification. To compare the EUS algorithm, the implementation of this GA based on the Darwinian theory of evolution was presented and used to improve classification alongside other well-known undersampling algorithms. This resulted in the improvement of the performance for most performance measures, primarily desired in IDS. Moreover, non-parametric statistical tests were presented and used to analyze the achieved results for any significant differences between the used undersampling methods. The implemented EUS algorithm showed a high resemblance to the RUS algorithm and showed no significant difference for the performance measures AUC and GM. The evaluation of the statistical tests also revealed that EUS performed worse when compared to other performance measures, such as F1-measure, accuracy, and false-positive rate.

However, several open ends remain in this thesis. The implementation of the EUS led to unforeseen issues during the evaluation that could not be addressed with the prevailing time constraints. The similarity between RUS and EUS indicates too much randomization during the evolutionary steps of the GA, which could be overcome by changing the implementation of several methods in the code. The components executing mutation and crossover are especially prone to randomization; therefore, methods to overcome this issue could be implemented, such as a reinitialization of the whole population or lowering the mutation rate. The improvement of the processing time required for undersampling would be another extension to this work.

A field that has not been covered in this thesis is the evaluation of additional datasets.

Especially datasets that are not in the field of intrusion detection but still extremely imbalanced, including medical or financial datasets. As long as there are imbalanced data distributions present, EUS would be a suitable method for increasing their classification performance. Furthermore, to increase the performance of EUS other supervised and unsupervised learning techniques such as clustering or outlier detection would be natural extensions to this work.

Concluding and briefly answering the formulated research question mentioned in Chapter 1, the EUS algorithm is a well-performing alternative to other commonly used under-sampling methods. The results presented in this thesis support this claim with several methods; however, the implementation of this thesis still needs some attention to fix the shortcomings mentioned above. The overwhelmingly positive results in the literature on EUS is why this algorithms' improvement will be a goal worth striving for in future projects.

Bibliography

[1] D. A. Cieslak, N. V. Chawla, and A. Striegel, "Combating imbalance in network intrusion data sets," in *2006 IEEE International Conference on Granular Computing*, pp. 732–737, 2006.

[2] M. Galar, A. Fernández, E. Barrenechea, *et al.*, "EUSBoost: Enhancing ensembles for highly imbalanced data-sets by evolutionary undersampling," *Pattern Recognition*, vol. 46, no. 12, pp. 3460–3471, 2013.

[3] B. Schneier, *Schneier on Security*. Wiley, 2009.

[4] J. Andress, *The Basics of Information Security: Understanding the Fundamentals of InfoSec in Theory and Practice*. Syngress basics series, Elsevier Science, 2014.

[5] Isaca, *CSX Cybersecurity Fundamentals Study Guide, 2nd Edition*. Information Systems Audit and Control Association, 2017.

[6] "Understanding difference between Cyber Security & Information Security - CISO Platform - CISO Platform." https://www.cisoplatform.com/profiles/blogs/understanding-difference-between-cyber-security-information. [Accessed: 2020-06-08].

[7] L. Wang, "Big Data in Intrusion Detection Systems and Intrusion Prevention Systems," *Journal of Computer Networks*, vol. 4, no. 1, pp. 48–55, 2017.

[8] A. Khraisat, I. Gondal, P. Vamplew, *et al.*, "Survey of intrusion detection systems: techniques, data sets and challenges," *Cybersecurity*, vol. 2, no. 1, pp. 1–22, 2019.

[9] Symantec, "Internet Security Threat Report ISTR." https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf, 2017. [Accessed: 2020-06-10].

[10] H. Liu and B. Lang, "Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey," *Applied Sciences*, vol. 9, no. 20, p. 4396, 2019.

[11] M. Friedman and A. Kandel, *Introduction to Pattern Recognition: Statistical, Structural, Neural, and Fuzzy Logic Approaches*. Series in machine perception and artificial intelligence, World Scientific, 1999.

[12] A. R. Webb and K. D. Copsey, *Statistical pattern recognition*. Wiley, 3rd ed., 2011.

[13] Z. Zhou, *Ensemble Methods: Foundations and Algorithms*. Chapman & Hall/CRC Data Mining and Knowledge Discovery Serie, Taylor & Francis, 2012.

[14] V. Dutt, V. Chadhury, and I. Khan, "Different approaches in pattern recognition," *IEEE Computer*, vol. 1, pp. 32–35, 2012.

[15] "Difference Between Supervised, Unsupervised, & Reinforcement Learning." https://blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning. [Accessed: 2020-06-10].

[16] R. Duda, P. Hart, and D. Stork, *Pattern Classification*. Wiley, 2012.

[17] C. M. Bishop, *Pattern Recogniton and Machine Learning*. Springer-Verlag New York, 2006.

[18] R. Polikar, "Ensemble based systems in decision making," *Circuits and Systems Magazine, IEEE*, vol. 6, no. 3, pp. 21–44, 2006.

[19] L. Mohammadpour, T. C. Ling, C. S. Liew, and C. Y. Chong, "A convolutional neural network for network intrusion detection system," *Proceedings of the Asia-Pacific Advanced Network*, vol. 46, pp. 50–55, 2018.

[20] C. Promper, D. Engel, and R. C. Green, "Anomaly detection in smart grids with imbalanced data methods," in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, 2017.

[21] V. López, A. Fernández, S. García, *et al.*, "An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics," *Information Sciences*, vol. 250, pp. 113–141, 2013.

[22] J. Ha and J.-S. Lee, "A New Under-Sampling Method Using Genetic Algorithm for Imbalanced Data Classification," in *Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication*, IMCOM '16, pp. 95:1–95:6, ACM, 2016.

[23] M. Kubat and S. Matwin, "Addressing the curse of imbalanced training sets: one-sided selection," in *Proc. 14th International Conference on Machine Learning*, pp. 179–186, Morgan Kaufmann, 1997.

[24] I. Triguero, M. Galar, D. Merino, J. Maillo, *et al.*, "Evolutionary undersampling for extremely imbalanced big data classification under apache spark," in *2016 IEEE Congress on Evolutionary Computation, CEC 2016*, pp. 640–647, Institute of Electrical and Electronics Engineers Inc., 2016.

[25] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[26] A. Fernández, S. García, M. Galar, *et al.*, *Learning from Imbalanced Data Sets*. Springer International Publishing, 2018.

[27] M. Galar, A. Fernandez, E. Barrenechea, *et al.*, "A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches," 2012.

[28] S. García and F. Herrera, "Evolutionary Undersampling for Classification with Imbalanced data sets: Proposals and Taxonomy," *Evolutionary Computation*, vol. 17, no. 3, pp. 275–306, 2009.

[29] Y. Sun, A. K. Wong, and M. S. Kamel, "Classification of imbalanced data: A review," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 23, no. 4, pp. 687–719, 2009.

[30] B. Krawczyk, M. Galar, and L. o. Jelen, "Evolutionary undersampling boosting for imbalanced classification of breast cancer malignancy," *Applied Soft Computing*, vol. 38, pp. 714–726, 2016.

[31] S. Shekarforoush, R. Green, and R. Dyer, "Classifying commit messages: A case study in resampling techniques," in *Proceedings of the International Joint Conference on Neural Networks*, vol. 2017-May, pp. 1273–1280, Institute of Electrical and Electronics Engineers Inc., 2017.

[32] C. Cernuda, *On the Relevance of Preprocessing in Predictive Maintenance for Dynamic Systems*, pp. 53–93. Springer International Publishing, 2019.

[33] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced data sets in machine learning," *Journal of Machine Learning Research*, vol. 18, no. 17, pp. 1–5, 2017.

[34] "imbalanced-learn 0.5.0 documentation." https://imbalanced-learn.readthedocs.io. [Accessed: 2020-06-15].

[35] A. N. Sloss and S. Gustafson, "2019 Evolutionary Algorithms Review," in *Genetic Programming Theory and Practice XVII*, pp. 307–344, Springer, Cham, 2020.

[36] J. H. . Holland, "Genetic Algorithms," *Scientific American*, vol. 267, no. 1, pp. 66–73, 1992.

[37] S. Mirjalili, "Genetic algorithm," in *Studies in Computational Intelligence*, vol. 780, pp. 43–55, Springer Verlag, 2019.

[38] D. J. Drown, T. M. Khoshgoftaar, and N. Seliya, "Evolutionary sampling and software quality modeling of high-assurance systems," *IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans*, vol. 39, no. 5, pp. 1097–1107, 2009.

[39] L. J. Eshelman, "The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination," in *Foundations of Genetic Algorithms* (G. J. E. Rawlins, ed.), vol. 1, pp. 265–283, Elsevier, 1991.

[40] S. Wang and X. Yao, "Relationships between diversity of classification ensembles and single-class performance measures," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 25, pp. 1 – 1, 2011.

[41] G. U. Yule, "On the association of attributes in statistics: With illustrations from the material of the childhood society," *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, vol. 194, pp. 257–319, 1900.

[42] G. Corder and D. Foreman, *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*. Wiley, 2011.

[43] G. Santafe, I. Inza, and J. A. Lozano, "Dealing with the evaluation of supervised classification algorithms," *Artificial Intelligence Review*, vol. 44, no. 4, pp. 467–508, 2015.

[44] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, p. 1–30, 2006.

[45] S. Garcia and F. Herrera, "An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons," *Journal of machine learning research*, vol. 9, no. Dec, pp. 2677–2694, 2008.

[46] N. Japkowicz and M. Shah, *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge University Press, 2011.

[47] "KDD Cup 1999 Data." http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. [Accessed: 2020-06-22].

[48] "1999 DARPA Intrusion Detection Evaluation data set." https://www.ll.mit.edu/r-d/datasets/1999-darpa-intrusion-detection-evaluation-dataset. [Accessed: 2020-06-22].

[49] M. V. Mahoney and P. K. Chan, "An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection," 2003.

[50] "The ADFA Intrusion Detection data sets." https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-IDS-datasets. [Accessed: 2020-06-22].

[51] "Ubuntu Operating System." https://ubuntu.com. [Accessed: 2020-06-22].

[52] "The Apache Software Foundation." http://www.apache.org. [Accessed: 2020-06-22].

[53] "PHP: Hypertext Preprocessor." https://www.php.net. [Accessed: 2020-06-22].

[54] "Tiki Wiki CMS Groupware." https://info.tiki.org. [Accessed: 2020-06-22].

[55] "MySQL." https://www.mysql.com. [Accessed: 2020-06-22].

[56] G. Creech and J. Hu, "Generation of a new IDS test data set: Time to retire the KDD collection," in *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 4487–4492, 2013.

[57] van Hauser, "Thc hydra." https://github.com/vanhauser-thc/thc-hydra, 2019.

[58] "Tiki Tikiwiki Cms/groupware: List of security vulnerabilities." https://www.cvedetails.com/vulnerability-list/vendor{_}id-12391/product{_}id-23390/Tiki-Tikiwiki-Cms-groupware.html. [Accessed: 2020-06-22].

[59] G. Creech, "Developing a high-accuracy cross platform host-based intrusion detection system capable of reliably detecting zero-day attacks," 2014.

[60] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting intrusions using system calls: Alternative data models," in *Proceedings - IEEE Symposium on Security and Privacy*, vol. 1999-Janua, pp. 133–145, Institute of Electrical and Electronics Engineers Inc., 1999.

[61] M. Xie and J. Hu, "Evaluating host-based anomaly detection systems: A preliminary analysis of ADFA-LD," in *Proceedings of the 2013 6th International Congress on Image and Signal Processing, CISP 2013*, vol. 3, pp. 1711–1716, 2013.

[62] J. VanderPlas, *Python Data Science Handbook: Essential Tools for Working with Data*. O'Reilly Media, Inc., 1st ed., 2016.

[63] "PyCharm." https://www.jetbrains.com/pycharm. [Accessed: 2020-06-23].

[64] W. McKinney, *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. O'Reilly Media, Inc., 2nd ed., 2017.

[65] "Anaconda." https://www.anaconda.com. [Accessed: 2020-06-23].

[66] "Python." https://www.python.org. [Accessed: 2020-06-23].

[67] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science and Engineering*, vol. 9, no. 3, pp. 99–104, 2007.

[68] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, p. 2825-2830, 2011.

[69] P. Virtanen, R. Gommers, T. E. Oliphant, *et al.*, "SciPy 1.0: fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, no. 3, pp. 261–272, 2020.

[70] S. Seabold and J. Perktold, "Statsmodels: Econometric and statistical modeling with python," in *9th Python in Science Conference*, 2010.

[71] "pickle — Python object serialization." https://docs.python.org/3/library/pickle.html. [Accessed: 2020-06-23].

[72] "API Reference — scikit-learn 0.23.1 documentation." https://scikit-learn.org/stable/modules/classes.html. [Accessed: 2020-06-26].

A. Additional Results

Listed hereafter are the complete classification results for all used undersampling methods and different performance measures (PM). All classifiers in Table A.1 were trained using the settings defined in Section 6.1.

Undersampler	Classifier	AUC	ACC	F1	GM	FNR	FPR
AllKNN	DTBagg	0.9326	0.9550	0.8341	0.9321	0.0973	0.0375
	DTBoost	0.9220	0.9437	0.7992	0.9215	0.1070	0.0490
	DT_d	0.9170	0.9392	0.7854	0.9166	0.1125	0.0534
	MLP	0.9377	0.9460	0.8114	0.9377	0.0733	0.0512
	MLP_d	0.9401	0.9500	0.8230	0.9400	0.0732	0.0466
	RForest	0.9368	0.9588	0.8468	0.9364	0.0926	0.0338
	k-NN	0.9365	0.9407	0.7973	0.9364	0.0692	0.0579
	k-NN_d	0.9270	0.9419	0.7965	0.9268	0.0928	0.0532
	linSVM	0.9273	0.9456	0.8063	0.9269	0.0973	0.0482
	rbfSVM	0.9275	0.9577	0.8403	0.9266	0.1129	0.0321
	Average	0.9304	0.9479	0.8140	0.9301	0.0928	0.0463
ENN	DTBagg	0.9334	0.9608	0.8515	0.9327	0.1032	0.0300
	DTBoost	0.9206	0.9487	0.8119	0.9199	0.1168	0.0419
	DT_d	0.9146	0.9437	0.7959	0.9138	0.1241	0.0466
	MLP	0.9390	0.9534	0.8319	0.9388	0.0802	0.0418
	MLP_d	0.9402	0.9551	0.8371	0.9400	0.0797	0.0399
	RForest	0.9362	0.9640	0.8622	0.9355	0.1008	0.0267
	k-NN	0.9377	0.9485	0.8179	0.9376	0.0767	0.0479
	k-NN_d	0.9275	0.9485	0.8142	0.9271	0.1005	0.0444
	linSVM	0.9247	0.9490	0.8142	0.9242	0.1076	0.0429
	rbfSVM	0.9246	0.9594	0.8443	0.9234	0.1218	0.0289
	Average	0.9299	0.9531	0.8281	0.9293	0.1011	0.0391
EUS	DTBagg	0.9294	0.9146	0.7359	0.9292	0.0509	0.0903
	DTBoost	0.9127	0.9034	0.7060	0.9126	0.0749	0.0997
	DT_d	0.9124	0.9042	0.7073	0.9124	0.0765	0.0986
	MLP	0.9287	0.9216	0.7499	0.9287	0.0617	0.0809
	MLP_d	0.9310	0.9143	0.7361	0.9307	0.0468	0.0912
	RForest	0.9337	0.9198	0.7486	0.9335	0.0478	0.0848
	k-NN	0.9238	0.8976	0.7013	0.9232	0.0412	0.1112
	k-NN_d	0.9102	0.8780	0.6620	0.9092	0.0467	0.1328
	linSVM	0.9246	0.9005	0.7067	0.9240	0.0433	0.1076
	rbfSVM	0.9259	0.9008	0.7081	0.9253	0.0405	0.1076
	Average	0.9232	0.9055	0.7162	0.9229	0.0530	0.1005
NCR	DTBagg	0.9343	0.9644	0.8629	0.9334	0.1058	0.0256
	DTBoost	0.9209	0.9528	0.8236	0.9199	0.1217	0.0365
	DT_d	0.9119	0.9455	0.7995	0.9108	0.1330	0.0433

Undersampler	Classifier	AUC	ACC	F1	GM	FNR	FPR
	MLP	0.9407	0.9573	0.8436	0.9404	0.0816	0.0371
	MLP_d	0.9405	0.9587	0.8477	0.9402	0.0838	0.0352
	RForest	0.9367	0.9665	0.8705	0.9359	0.1030	0.0235
	k-NN	0.9419	0.9533	0.8327	0.9418	0.0733	0.0429
	k-NN_d	0.9317	0.9524	0.8264	0.9313	0.0958	0.0407
	linSVM	0.9206	0.9498	0.8149	0.9198	0.1184	0.0404
	rbfSVM	0.9215	0.9604	0.8464	0.9201	0.1304	0.0266
	Average	0.9301	0.9561	0.8368	0.9294	0.1047	0.0352
OSS	DTBagg	0.9256	0.9711	0.8823	0.9236	0.1352	0.0137
	DTBoost	0.9171	0.9663	0.8636	0.9148	0.1485	0.0173
	DT_d	0.9007	0.9521	0.8133	0.8981	0.1680	0.0307
	MLP	0.9298	0.9668	0.8692	0.9285	0.1195	0.0208
	MLP_d	0.9305	0.9673	0.8712	0.9292	0.1186	0.0204
	RForest	0.9277	0.9728	0.8887	0.9258	0.1324	0.0122
	k-NN	0.9313	0.9639	0.8605	0.9302	0.1123	0.0252
	k-NN_d	0.9226	0.9622	0.8522	0.9211	0.1302	0.0246
	linSVM	0.8998	0.9534	0.8167	0.8970	0.1716	0.0287
	rbfSVM	0.9019	0.9620	0.8443	0.8983	0.1783	0.0179
	Average	0.9187	0.9638	0.8562	0.9167	0.1415	0.0211
RUS	DTBagg	0.9313	0.9171	0.7417	0.9311	0.0498	0.0877
	DTBoost	0.9164	0.9083	0.7171	0.9163	0.0729	0.0944
	DT_d	0.9131	0.9043	0.7078	0.9130	0.0751	0.0987
	MLP	0.9289	0.9239	0.7552	0.9289	0.0644	0.0777
	MLP_d	0.9330	0.9172	0.7430	0.9328	0.0459	0.0880
	RForest	0.9358	0.9225	0.7551	0.9356	0.0464	0.0820
	k-NN	0.9255	0.8998	0.7061	0.9249	0.0403	0.1087
	k-NN_d	0.9118	0.8796	0.6654	0.9108	0.0453	0.1311
	linSVM	0.9257	0.9026	0.7112	0.9251	0.0436	0.1051
	rbfSVM	0.9263	0.9023	0.7109	0.9257	0.0416	0.1058
	Average	0.9248	0.9078	0.7213	0.9244	0.0525	0.0979
RENN	DTBagg	0.9337	0.9514	0.8245	0.9334	0.0901	0.0426
	DTBoost	0.9189	0.9375	0.7821	0.9186	0.1059	0.0562
	DT_d	0.9172	0.9355	0.7764	0.9169	0.1073	0.0583
	MLP	0.9369	0.9408	0.7979	0.9369	0.0682	0.0579
	MLP_d	0.9409	0.9452	0.8104	0.9409	0.0648	0.0534
	RForest	0.9371	0.9554	0.8368	0.9368	0.0872	0.0385
	k-NN	0.9360	0.9345	0.7822	0.9360	0.0621	0.0660
	k-NN_d	0.9279	0.9366	0.7837	0.9278	0.0838	0.0605
	linSVM	0.9281	0.9422	0.7978	0.9279	0.0907	0.0530
	rbfSVM	0.9314	0.9563	0.8374	0.9308	0.1018	0.0354
	Average	0.9308	0.9435	0.8029	0.9306	0.0862	0.0522

Undersampler	Classifier	AUC	ACC	F1	GM	FNR	FPR
TL	DTBagg	0.9247	0.9709	0.8814	0.9226	0.1370	0.0137
	DTBoost	0.9169	0.9667	0.8649	0.9145	0.1495	0.0167
	DT_d	0.8998	0.9527	0.8146	0.8970	0.1708	0.0296
	MLP	0.9302	0.9671	0.8703	0.9289	0.1191	0.0206
	MLP_d	0.9307	0.9680	0.8736	0.9293	0.1192	0.0195
	RForest	0.9278	0.9730	0.8896	0.9259	0.1325	0.0119
	k-NN	0.9325	0.9659	0.8672	0.9314	0.1122	0.0229
	k-NN_d	0.9236	0.9625	0.8536	0.9222	0.1283	0.0245
	linSVM	0.9000	0.9536	0.8175	0.8971	0.1716	0.0284
	rbfSVM	0.9009	0.9619	0.8437	0.8972	0.1806	0.0176
	Average	0.9187	0.9642	0.8576	0.9166	0.1421	0.0205

Table A.1.: Full table of tested undersamplers and classifier results

In the evaluation part of this thesis only the Friedman Test results for the performance metric AUC have been presented. Table A.2 shows the results of the remaining performance measures.

PM	Undersampler	Friedman score	Friedman p-value	H0 rejected
AUC	AllKNN	159.273	1.05e-23	True
	ENN	162.589	2.15e-24	True
	EUS	158.356	1.63e-23	True
	NCR	164.215	9.88e-25	True
	OSS	154.287	1.14e-22	True
	RUS	153.284	1.84e-22	True
	RENN	163.440	1.43e-24	True
	TL	155.356	6.84e-23	True
ACC	AllKNN	167.321	2.23e-25	True
	ENN	166.066	4.07e-25	True
	EUS	164.302	9.47e-25	True
	NCR	171.994	2.37e-26	True
	OSS	166.429	3.42e-25	True
	RUS	162.622	2.12-24	True
	RENN	169.375	8.33e-26	True
	TL	169.287	8.69e-26	True
F1	AllKNN	168.589	1.21e-25	True
	ENN	166.920	2.70e-25	True
	EUS	157.549	2.40e-23	True
	NCR	172.178	2.17e-26	True
	OSS	163.942	1.13e-24	True
	RUS	158.716	1.37e-23	True
	RENN	169.004	9.95e-26	True

PM	Undersampler	Friedman score	Friedman p-value	H0 rejected
	TL	167.946	1.65e-25	True
FNR	AllKNN	157.500	2.45e-23	True
	ENN	161.493	3.64e-24	True
	EUS	143.667	1.80e-20	True
	NCR	170.330	5.27e-26	True
	OSS	163.082	1.70e-24	True
	RUS	137.378	3.59e-19	True
	RENN	165.430	5.52e-25	True
	TL	164.463	8.77e-25	True
FPR	AllKN	163.760	1.23e-24	True
	ENN	161.103	4.38e-24	True
	EUS	168.480	1.28e-25	True
	NCR	168.315	1.38e-25	True
	OSS	170.484	4.89e-26	True
	RUS	166.164	3.88e-25	True
	RENN	167.496	2.05e-25	True
	TL	173.232	1.31e-26	True
GM	AllKNN	159.426	9.78e-24	True
	ENN	162.349	2.41e-24	True
	EUS	158.509	1.52e-23	True
	NCR	164.727	7.73e-25	True
	OSS	154.189	1.19e-22	True
	RUS	155.095	7.75e-23	True
	RENN	162.655	2.08e-24	True
	TL	156.753	3.51e-23	True

Table A.2.: Full table of Friedman Test results

The Wilcoxon Ranked-Sum Test results for all performance measures are shown in Table A.3.

PM	Comparison	Wilcoxon Rank	Wilcoxon p-value	H0 rejected
AUC	EUS vs. AllKNN	27	0.0442	True
	EUS vs. ENN	30	0.0699	False
	EUS vs. NCR	31	0.0807	False
	EUS vs. OSS	42.5	0.2979	False
	EUS vs. RUS	43	0.3108	False
	EUS vs. RENN	24	0.0267	True
	EUS vs. TL	42.5	0.2980	False
ACC	EUS vs. AllKNN	0	0.0001	True
	EUS vs. ENN	0	0.0001	True
	EUS vs. NCR	0	0.0001	True

PM	Comparison	Wilcoxon Rank	Wilcoxon p-value	H0 rejected
	EUS vs. OSS	0	0.0001	True
	EUS vs. RUS	41	0.2599	False
	EUS vs. RENN	0	0.0001	True
	EUS vs. TL	0	0.0001	True
F1	EUS vs. AllKNN	0	0.0001	True
	EUS vs. ENN	0	0.0001	True
	EUS vs. NCR	0	0.0001	True
	EUS vs. OSS	0	0.0001	True
	EUS vs. RUS	37	0.1718	False
	EUS vs. RENN	0	0.0001	True
	EUS vs. TL	0	0.0001	True
FNR	EUS vs. AllKNN	6	0.0005	True
	EUS vs. ENN	0.5	0.0001	True
	EUS vs. NCR	2	0.0002	True
	EUS vs. OSS	0	0.0001	True
	EUS vs. RUS	44.5	0.3525	False
	EUS vs. RENN	6.5	0.0006	True
	EUS vs. TL	0	0.0001	True
FÜR	EUS vs. AllKNN	0	0.0001	True
	EUS vs. ENN	0	0.0001	True
	EUS vs. NCR	0	0.0001	True
	EUS vs. OSS	0	0.0001	True
	EUS vs. RUS	42.5	0.2981	False
	EUS vs. RENN	0	0.0001	True
	EUS vs. TL	0	0.0001	True
GM	EUS vs. AllKNN	27	0.0441	True
	EUS vs. ENN	31	0.0807	False
	EUS vs. NCR	32.5	0.0988	False
	EUS vs. OSS	38.5	0.2018	False
	EUS vs. RUS	42	0.2846	False
	EUS vs. RENN	24.5	0.0292	True
	EUS vs. TL	38	0.1909	False

Table A.3.: Full table of all Wilcoxon Ranked-Sum Test pairwise comparisons

Table A.4 shows the obtained results of the Holm p-value correction method for all calculated performance measures.

PM	Comparison	H0 rejected	Corrected p-value	AVG Difference
AUC	EUS vs. AllKNN	False	0.4926	-0.0066
	EUS vs. ENN	False	0.5621	0.0000
	EUS vs. NCR	False	0.6037	-0.0068
	EUS vs. OSS	False	10.000	0.0045
	EUS vs. RUS	False	10.000	-0.0015
	EUS vs. RENN	False	0.3456	-0.0076
	EUS vs. TL	False	10.000	0.0045
ACC	EUS vs. AllKNN	True	0.0011	-0.0476
	EUS vs. ENN	True	0.0011	0.0000
	EUS vs. NCR	True	0.0011	-0.0506
	EUS vs. OSS	True	0.0011	-0.0583
	EUS vs. RUS	False	0.4963	-0.0023
	EUS vs. RENN	True	0.0011	-0.0381
	EUS vs. TL	True	0.0011	-0.0588
F1	EUS vs. AllKNN	True	0.0011	-0.1119
	EUS vs. ENN	True	0.0011	0.0000
	EUS vs. NCR	True	0.0011	-0.1206
	EUS vs. OSS	True	0.0011	-0.1400
	EUS vs. RUS	False	0.3258	-0.0051
	EUS vs. RENN	True	0.0011	-0.0867
	EUS vs. TL	True	0.0011	-0.1414
FNR	EUS vs. AllKNN	True	0.0026	-0.0481
	EUS vs. ENN	True	0.0011	0.0000
	EUS vs. NCR	True	0.0011	-0.0516
	EUS vs. OSS	True	0.0011	-0.0884
	EUS vs. RUS	False	0.7055	0.0005
	EUS vs. RENN	True	0.0026	-0.0331
	EUS vs. TL	True	0.0011	-0.0890
FPR	EUS vs. AllKNN	True	0.0011	0.0613
	EUS vs. ENN	True	0.0011	0.0000
	EUS vs. NCR	True	0.0011	0.0653
	EUS vs. OSS	True	0.0011	0.0793
	EUS vs. RUS	False	0.5967	0.0025
	EUS vs. RENN	True	0.0011	0.0483
	EUS vs. TL	True	0.0011	0.0799
GM	EUS vs. AllKNN	False	0.4926	-0.0064
	EUS vs. ENN	False	0.6529	0.0000
	EUS vs. NCR	False	0.6945	-0.0065
	EUS vs. OSS	False	10.000	0.0062

PM	Comparison	H0 rejected	Corrected p-value	AVG Difference
	EUS vs. RUS	False	10.000	-0.0015
	EUS vs. RENN	False	0.3456	-0.0077
	EUS vs. TL	False	10.000	0.0063

Table A.4.: Full table of Holm method results with corrected p-values