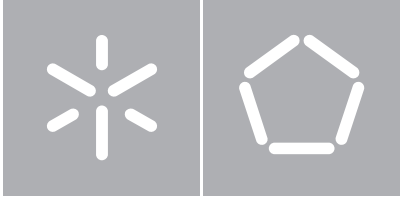
Universidade do Minho
Escola de Engenharia

Ricardo Manuel Andrade Costeira

Detecção Inteligente de Fugas de Informação por Análise Comportamental

Março de 2016



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Ricardo Manuel Andrade Costeira

Detecção Inteligente de Fugas de Informação por Análise Comportamental

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho realizado sob orientação de
Professor Henrique Santos
Engenheiro Bernardo Patrão

Março de 2016

Anexo 3

DECLARAÇÃO

Nome

_____

Endereço electrónico: _____ Telefone: _____ / _____

Número do Bilhete de Identidade: _____

Título dissertação □/tese □

_____

_____

_____

Orientador(es):

_____

_____ Ano de conclusão: _____

Designação do Mestrado ou do Ramo de Conhecimento do Doutoramento:

_____

Nos exemplares das teses de doutoramento ou de mestrado ou de outros trabalhos entregues para prestação de provas públicas nas universidades ou outros estabelecimentos de ensino, e dos quais é obrigatoriamente enviado um exemplar para depósito legal na Biblioteca Nacional e, pelo menos outro para a biblioteca da universidade respectiva, deve constar uma das seguintes declarações:

1. É AUTORIZADA A REPRODUÇÃO INTEGRAL DESTA TESE/TRABALHO APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

2. É AUTORIZADA A REPRODUÇÃO PARCIAL DESTA TESE/TRABALHO (indicar, caso tal seja necessário, nº máximo de páginas, ilustrações, gráficos, etc.), APENAS PARA EFEITOS DE INVESTIGAÇÃO, , MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

3. DE ACORDO COM A LEGISLAÇÃO EM VIGOR, NÃO É PERMITIDA A REPRODUÇÃO DE QUALQUER PARTE DESTA TESE/TRABALHO

Universidade do Minho, ___/___/_____

Assinatura: _____

Acknowledgements

I would like to express my gratitude to my advisor, professor Henrique Santos, for his enlightening and invaluable guidance through these past two years. His suggestions and insight about the subject provided for a solid ground from which to start the investigation, while at the same time allowed me to consistently build everything up towards the end.

I would also like to thank *Watchful Software* for the opportunity to work with them, and *iTGrow* for partially supporting my research. Many thanks to everyone at *Watchful Software's* engineering team for being so receptive, helpful, kind, and making me feel at home. Special thanks to Paulo, Daniel and Bernardo for their interest, suggestions and guidance - which was vital for my work - and to everyone in the team for creating this amazing and relaxed, yet productive, atmosphere at the work place.

Of course, I also have to thank my parents, my sister and my closest friends - Emanuel, Marco, Marina and Luísa - for their constant support and for putting up with me this whole time, and my other "family", back at the University - Fábio, Cris, João, André, David, Carla, Fernando, and everyone else - for making these last seven years some of the best years of my life (so far!). Every single one of these people contributed into shape me to what I am today.

Lastly, I have to thank Inês, who experienced all the joy and pain of developing this dissertation almost as much as me. Thank you so much for all your love, support and especially your patience. Thank you for all the encouraging words, and for always believing in me. You alone gave me all the strength I needed in order to finish the writing. And sorry for still having the habit of biting my nails.

P.S.: Thank you Ludovico Einaudi, Kimbra, Hanz Zimmer, Two Steps From hell, Immediate Music, Globus, Muse, the xx, noiserv and all the other bands, groups and individual artists that helped me focused while both developing and writing.

UNIVERSITY OF MINHO

Abstract

School of Engineering

Informatics Department

Master of Science Degree

Inteligent Data Leak Detection Through Behavioural Analysis

by Ricardo Costeira

The information that a company possesses is one of its most valuable assets. This information is nowadays digitally managed, which is the reason for the exponential increase in security breaches, where information is defiled or even stolen. Seeking to solve this problem, *Watchful Software* developed a product, **RightsWATCH**, that allows for an organization to protect and watch over its information.

By monitoring what happens to information, *RightsWATCH* provides, in case of an incident, the means to undertake a very complete *post-mortem* analysis. Nevertheless, by the time this analysis is complete, it might have been hours (or days) since the incident occurred. To make matters worse, nowadays most threats actually come from the inside of the company. That being said, this dissertation defines as its main objective the need to understand if it is possible to detect data leaks in an intelligent way, through a real time analysis of the user's behaviour while he handles the classified information. This possibility was indeed confirmed through an investigation comprising experiences with real world use cases and a variety of data preparation and data analysis techniques.

UNIVERSIDADE DO MINHO

Resumo

Escola de Engenharia

Departmento de Informática

Grau de Mestre

Detecção Inteligente de Fugas de Informação por Análise Comportamental

por Ricardo Costeira

A informação gerada por uma organização é um dos seus bens mais valiosos. Actualmente, essa informação é normalmente gerida de forma digital, razão pela qual se tem verificado um aumento exponencial de incidentes de segurança, onde esta informação é adulterada, ou até mesmo roubada. Com vista ao combate a este problema, a *Watchful Software* desenvolveu um produto, o **RightsWATCH**, que permite a uma organização proteger e monitorizar a sua informação.

Ao monitorizar o que acontece à informação, o *RightsWATCH* permite, no caso de incidente, que seja efectuada uma análise *post-mortem* bastante completa. Não obstante, quando essa análise é feita, podem já ter passado horas (ou dias) desde a ocorrência do incidente. Para piorar a situação, a maior parte das ameaças de hoje em dia são internas. Assim sendo, esta dissertação tem como questão central perceber se existe a possibilidade de detectar, de forma inteligente e em tempo real, fugas de informação através da análise do comportamento dos utilizadores aquando do manuseamento da informação protegida. Esta possibilidade foi de facto confirmada através de um trabalho de investigação que envolveu experiências com casos de uso reais e a aplicação de várias técnicas de preparação e análise de dados.

Contents

List of Figures

Abbreviations

IDS	**I**ntrusion **D**etection **S**ystem
IDPS	**I**ntrusion **D**etection and **P**revention **S**ystem
HIDS	**H**ost-based **I**ntrusion **D**etection **S**ystem
NIDS	**N**etwork-based **I**ntrusion **D**etection **S**ystem
NIC	**N**etwork **I**nterface **C**ard
DMZ	**D**e**M**ilitarized **Z**one
IDWG	**I**ntrusion **D**etection Exchange Format **W**orking **G**roup
IETF	**I**nternet **E**ngineering **T**ask **F**orce
RFC	**R**equest **F**or **C**omments
FTP	**F**ile **T**ransfer **P**rotocol
ADS	**A**nomaly **D**etection **S**ystem
MLE	**M**aximum **L**ikelihood **E**stimates
PDF	**P**robability **D**istribution **F**unction
KDD	**K**nowledge **D**iscovery in **D**atabases
PCA	**P**rincipal **C**omponent **A**nalysis
SVM	**S**upport **V**ector **M**achine
RBF	**R**adial **B**asis **F**unction
AD RMS	**A**ctive **D**irectory **R**ights **M**anagement **S**ervices
DBMS	**D**ata**B**ase **M**anagement **S**ystem
C-SVC	**C**-**S**upport **V**ector **C**lassification
BYOD	**B**ring **Y**our **O**wn **D**evice

Chapter 1

Introduction

1.1 The Problem

1.1.1 The Value of Information

The evolution of technology and information systems has opened a brave new world of possibilities for companies and organizations all over the globe. The need for each individual company to invest in rooms, or even warehouses of file cabinets and paper documents is no more, since information can now be stored on some data centre on any part of the world - although some companies still choose to have their own data centres, which translates into extra costs, having full control over it still pays off. The advent of the Internet brought cloud storage, which makes information accessible wherever the user is and whenever the user wants, as long as he can connect to the service. The Internet is also responsible for providing the largest information repository of the world, since it provides the means for anyone to share the information they want, as well as to search for any kind of information: It even allows for the creation of mechanisms designed to retrieve information about other users or entities.

These breakthroughs, along with many others, provided easiness of creation, access to and use of information, and gradually made an impact on a company's *modus operandi*. In fact, if a company is in possession of the right information, it can develop its products or services in a way that is best suited for the target audience's needs, undoubtedly gaining some advantage

over competitors. Thus, in this day and age, and apart from the collaborators themselves, **information is the most important asset of a company**. Obviously, this also creates the need to protect it.

This need gets even more relevant as the value of corporate information is usually directly proportional to its level of confidentiality: The more exclusive the access to information, the more can a company profit from it, meaning that information can lose all its value if someone else gets a hold of it. It is common for this to happen, as organizations often suffer from data leak. To prevent data leak, organizations have invested billions in information security technologies like firewalls, antivirus and intrusion prevention services, among others. Regardless, although relevant, these measures are by far insufficient, as corporate information keeps going out of the secure perimeter: employees come, and employees go, and as they leave the company they usually have the freedom to take sensitive corporate information, whether they do it on purpose or not. An employee might want to work out of the company, which means that he/she might carry around classified information on whatever device(s) (notebook, smartphone, tablet, ...) he/she owns. Situations like these create breaches on the network, giving birth to data leaks - devices can be lost, stolen, or even hacked; careless employees can mistakenly store the information on the wrong place, or send it to the wrong person; employees with less noble intentions might sell the information to the competition. According to a research study from Ponemon Institute (Ponemon Institute), performed over nine countries, in 2012, regarding 227 organizations from sixteen different industry sectors, seventy-two percent of data leak cases occur due to human error or malicious attacks. This study also claims that these data leaks costed 3 890 623,68 US dollars to the United States, and 3 472 979,76 US dollars to Germany. These are the highest values reported by the study though - the lowest belongs to India, with 803 378,88 US dollars, which is still a considerable amount (Ponemon Institute).

It is by now clear that conventional methods are not enough, due to the granularity of the protection they offer being so coarse. Instead, why keep wasting money protecting the network, when protecting the information itself would be so much more reliable? This is called **data-centric information protection** (Bayuk, 2009). The problem here though, is that data-centric information protection requires the user to have knowledge on data encryption, data tokenization or whatever method the company decides to use, which is highly impractical - imagine a marketing manager having to call an IT person whenever he wanted to protect

information. So, is there an alternative way to address this extremely inconvenient issue?

1.1.2 Data-Centric Information Protection

Data-centric security is a concept that stresses the security of data and information itself, rather than securing networks, workstations, applications, etc. What companies and organisations are starting to understand nowadays is that a data-centric security solution is the only current way to ensure that their precious data is safe. Currently, there are a few software tools capable of data-centric protection. One of these is called *RightsWATCH*.

RightsWATCH is a product developed by *Watchful Software*, an information security company founded in Coimbra, spin-off and member of the renowned **Critical Group**. *RightsWATCH* focuses on protecting the information itself - Emails, *Microsoft Office* documents, any type of document - on enterprise environments, where confidentiality of information is a relevant and competitive factor. The information is classified (and encrypted, if needed) according to a set of security levels defined by the company. The company's collaborators are granted clearance levels given their status, which allows them to access certain types of information. For instance, a common collaborator at *Watchful Software* has full access to *Public* and *Internal* information, but only has reading rights for *Confidential* information. This means that the collaborator has very limited power over the information - he cannot print, copy, print screen, forward (in case of an email), among other actions. When it comes to information classified as *Secret* however, this collaborator cannot even read it. A company can have as many levels as it needs, and it can aggregate them on what is called a scope. Scopes are useful for managing levels according to their purpose. For instance, a company can have a scope named *Finances* and another one called *Partners*, with completely different levels on each one.

The protection that *RightsWATCH* offers is embedded in the document, meaning that it stays protected wherever it is being used, sent, lost on a device, offline, etc. The classification mechanisms are also embedded on the information handling tools - *Microsoft Office*, *Sharepoint*, handheld devices, webmail - in order for their usage to remain seamless to the user.

So far, the problem seems solved. However *RightsWATCH*, in spite of doing an excellent job with information protection, does not account for one ever-changing, uncontrollable variable:

people. There are incidents caused by a company's collaborators that *RightsWATCH* simply cannot prevent. For instance, an employee can send unprotected information to someone outside of the company, because he/she simply forgot to set the security level. A disgruntled employee with the right credentials can decrypt and remove classification marks from confidential or even secret information - of course, depending on the company's security settings -, and send it to the company's competitors. If any of these cases do happen, or any other that is not supposed to, *RightsWATCH* provides a logging and monitoring interface meant to minimize the damage by allowing the system administrator to conduct a *post-mortem* analysis. This analysis consists of the inspection of the event logs generated by user interaction with the information that was leaked. Events such as "*was it forwarded*", "*was the level of classification changed*", "*was it encrypted*" are examined. Digging through the enormous amount of logs is a time-consuming and difficult task. Indeed, it is extremely hard for a human to look at the data and extract useful conclusions, even though the logs are in a readable format. Therefore, even if this log analysis process reveals what really happened, it might be too late, and the value of the information may have been compromised already. With this in mind, a new and exciting idea begins to take form: In order to tighten even more the data-centric security and protection it provides, *RightsWATCH* needs a way to understand the intentions of the company's collaborators, in the interest of preventing data leak even before it happens.

1.2 Goals

RightsWATCH's monitoring framework captures a large array of different events. Every time a user performs some kind of action on an asset protected by *RightsWATCH*, an event is generated, captured and stored in a database. In a sense, all the events regarding the actions of a certain user, when grouped together, can be interpreted as that user's way of interacting with the information or, in a more convenient way, his behaviour towards the company's corporate or sensitive information.

That being said, it can now be stated that the main goal of this dissertation is to investigate the viability of developing a system that can: 1) gather all the logs generated from *RightsWATCH* and create a model from them that can represent the user's behaviour towards the protected data and 2) make use of that model to output a decision regarding the nature of new behaviour

- if it is consistent with the user's behaviour so far or if it is a new, unknown (thus abnormal) type of behaviour. This system should provide results in real time, or closely enough. Developing the system to be the most generic as possible is also a goal. Notwithstanding, concerning the scope of the investigation, the sole function of the system will be to extract useful patterns from event logs, in order to output consistent and trustworthy decisions regarding the user's intentions. Why patterns? Because patterns can be one of the most revealing and important information types. Patterns let us know how things work - farmers seek patterns in crop growth; physicists seek patterns on the interactions between objects and matter, in order to understand how the world works and to enclose this understanding in theories that can be further used to predict what will happen in similar or new situations; politicians seek patterns in voters' opinions. Through pattern usage it is possible to reduce the dimension of a search domain, by grouping values associated to common facts and building easy to analyze models. Intelligently analysed data is a valuable resource, as it can lead to new insights and, in cases like this, to competitive advantages (Witten et al., 2011). This leads to the other goal of this investigation, which is to develop the architecture of this system in such a way that it can be easily integrated with *RightsWATCH*. This also includes the subgoal of implementing the system in a more or less generic way with respect to the access to data sources: It has to be able to feed itself from different data sources and gather all the data together with little or no extra effort.

1.3 Research Methodology

The investigation that supported this dissertation followed a *Design Science* approach, focusing on the seven guidelines proposed in (Hevner et al., 2004): Design as an artefact, problem relevance, design evaluation, research contributions, research rigour, design as a search process and research communication.

Design as an Artefact

This dissertation presents a functional anomaly detection framework. The system can be adapted to access different data sources, and it will either allow customisation by the manual selection of detection algorithms and/or parameters or find them by itself, which provides the possibility of applying the framework to different domains.

Problem Relevance

Anomaly detection is extremely important in safety critical environments and in sectors where the existence of anomalies can cost more than what the entity deems as acceptable. Apart from this, it is also a hard problem to solve, mainly due to the inability of reproducing every possible anomaly that might exist in a given domain. The developed framework is meant to provide a set of tools and functionalities that will greatly aid any system where it is integrated in tackling the anomaly detection problem.

Design Evaluation

The framework was thoroughly evaluated by following the five design evaluation methods mentioned in (Hevner et al., 2004). The artefact was tested by using *Watchful Software* as a case study. The different scenarios of *RightsWATCH* misuse already described provide the needed description of the framework's utility. Its architectural design was analysed in terms of how it complements *RightsWATCH*, and both structural and functional tests were conducted to guarantee the correct workflow of the framework and that the end result is the one expected. The classifiers' creation was studied in a controlled environment, and these were evaluated by the use of information retrieval metrics such as false positive and false negative rates, which are then computed to output more refined measures like precision and recall.

Research Contributions

The main contribution of this research is the design artefact itself. Existing knowledge is applied in order to solve a very specific problem that, until now, had no acceptable solution.

Research Rigour

The developed anomaly detection framework was deeply tested with real datasets, derived from information pertaining to the use of *RightsWATCH* by *Watchful Software*'s own employees. The tests were done regarding both user privacy and framework performance and accuracy. Accepted and well documented methods such as *k-fold cross validation* where used. Also, every single test was repeated more than once to account for any type of variation, which is minimal as soon as the classifiers are formed. The decisions that were made along the way are backed up by tests and resulting conclusions.

Design as a Search Process

The theory behind anomaly detection itself was studied, such as data and anomaly types. After the definition and discussion of a myriad of anomaly detection methods, the one which seems more appropriate for the problem at hands was chosen and the search for the best features available was conducted. By testing these features with the initial dataset, it was concluded that the dataset had to be changed in order to achieve better results. The dataset was completely reshaped, and new features were created from the initial ones. With this new dataset, not only the results obtained were slightly improved and more robust, but also the dataset became easier to interpret to a human mind.

Research Communication

Apart from this dissertation, the conducted research is also documented on a paper currently undergoing a submission process for an international conference called "*CYBER 2016, The*

First International Conference on Cyber-Technologies and Cyber-Systems".

1.4 Report Structure

This report is structured into five main chapters, the first of which is this introduction. The second chapter provides for an extensive investigation regarding the state-of-the-art technologies and methodologies used in intrusion detection systems. Architectural designs and algorithmic approaches are analysed and compared. At the end of this chapter, some of the most popular and relevant systems are discussed.

Chapter 3 concerns the proposed architecture for the framework, and chapter 4 regards all the verification and validation processes that were accomplished in order to assess the framework's performance. Results are analysed and discussed as they are obtained. Finally, the fifth and last chapter documents the final conclusions and future plans for the framework.

Chapter 2

Intrusion Detection Systems

2.1 Definition

An intrusion detection system, also known as intrusion detection *and prevention* system (or **IDPS**) when it has preventive or counter-attack measures against intrusions, is a tool capable of detecting possible security breaches on a system, by gathering and analysing information that it was specifically designed to understand. It can be designed to work with a wide range of information, such as event logs from different sources (firewalls, OSs, etc.), application usage data, keyboard inputs, or network data packets. Although the main objective of an *IDS* is, as the name suggests, to detect intrusions, these systems are often more complex. According to (Venkaiah et al., 2010), an intrusion detection system should provide three security functions: it has to *monitor* the computer or network, to *detect* possible threats and to *respond* to the possible intrusion. This response is ruled by a set of policies that will dictate what should the system do, depending on the triggered event - it can either issue a warning for the system administrator, or respond automatically by, for instance, blocking a user or launching some application or script.

An *IDS* does not guarantee full protection, but it does create what can be called a second line of defence, complementing other security control services such as access control or firewalls. Similarly to some of these services, *IDS*s operate in an almost seamless way, integrating themselves smoothly into the user's daily workflow.

As it seems, the concept of "intrusion detection system" was first used on the early 80's, in a seminal paper by Anderson (Anderson, 1980). At this time, *IDS*s were focused on single machines - Jones and Sielken (Jones and Sielken, 2000) called this the **first generation** of intrusion detection. Denning (Denning, 1987) introduced a new intrusion detection model called "intrusion-detection expert system" (*expert systems* are described on section 2.4.1), meant to detect a wider range of security violations through more sophisticated processing, more behaviour patterns and, more importantly, real-time alerts - the **second generation** of *IDS*s had arrived, and it was able to expand from a single computer to a distributed system. As time passed on, and with the growth of the Internet and communication technologies, Heberlein (Heberlein et al., 1990) brought *Network Security Monitor*, or *NSM* (section 2.5.4) and the **third** (and current) **generation** of intrusion detection systems, whose main focus is on networks.

Porras and Valdes (Porras and Valdes, 1998) quantitatively evaluate the success and failure of an intrusion detection system with three different measurements: **accuracy**, **performance**, and **completeness**. The first two are relatively easy to understand, and to measure as well - accuracy regards the *IDS*'s ability to correctly flag as anomalous or intrusive any malicious action, and performance has to do with the rate at which the *IDS* can analyse and process audit events. Completeness, on the other hand, is the measure of how well can the system detect an attack - incompleteness occurs when the *IDS* fails to detect an attack. Clearly this evaluation criteria will have some degree of uncertainty, as it is indeed impossible to have a global awareness of attacks or privilege abuse.

In addition to these metrics, Debar (Debar et al., 1999) introduced two more concepts: **fault tolerance** and **timeliness**. Intrusion detection systems should be designed to be fault tolerant, so that they are attack resistant, especially regarding denial of service - this becomes even more relevant considering that the operating system on which the *IDS* is installed, or even the hardware where it runs, may be vulnerable to attacks. The *timeliness* dimension is similar to performance, the difference being that it also includes the time required by the *IDS* to propagate and react to information - the intrusion detection system has to complete and send its analysis to the security officer as soon as possible, to both minimize the damage and prevent the attacker from overthrowing the analysis or the *IDS* itself.

Axelsson (Axelsson, 1998), based on the evaluation of several intrusion detection systems that

existed at the time, developed a generalized or, as the author states, "typical" model of an intrusion detection system. This model is shown in figure 2.1: the solid arrows represent data/control flow, while dotted arrows indicate a possible response to intrusion activity. Note that although several different databases are represented, this only means that the corresponding data is stored, and not that there actually is a need for several independent databases. Axelsson described each module as follows:

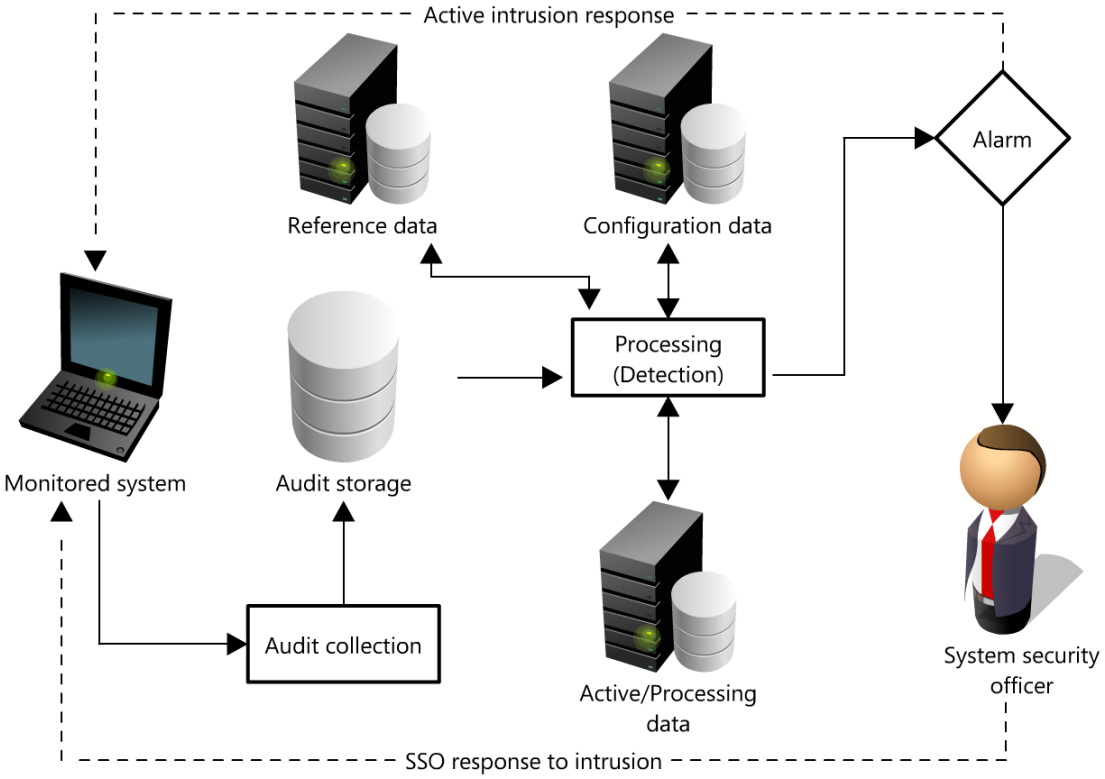


FIGURE 2.1: Generic model of an intrusion detection system.

Audit collection - Aims to collect information from where the system will extract knowledge to support its decisions (as referenced earlier, the collected data can come from a wide variety of sources). Typically, the term **sensor** is applied to systems that monitor networks, while host based technologies tend to use the term **agent** (Scarfone and Mell, 2007). Network based and host based technologies will be tackled on a later section.

Audit storage - The audit data has to be stored for further reference, and for long enough to allow for its processing. Since its volume is often exceedingly large, the storage space

can easily get overwhelmed. Because of this, intrusion detection can be considered as a problem in audit data reduction.

Processing - This module is the core of the *IDS*. This is where one or more intrusion detection algorithms are executed to evaluate the available information in search of suspicious behaviour. Over the years, these algorithms have been usually classified into three different categories: **anomaly detection** (also called **behaviour based detection**), **misuse detection** (also known as **signature based** or **knowledge based detection**), and **hybrid detection**. These techniques will be discussed in section 2.4.

Configuration data - This is the module where all the information regarding the way the intrusion detection should be carried on is stored. How to collect the audit data, how should an intrusion be dealt with, which threshold value distinguishes intrusions from normal behaviour, etc. Due to its nature, this is also the most sensitive module, as access to the information stored here would give an intruder the means to fool the system, thus being able to succeed in going by undetected.

Reference data - Concerns all the information used as a term of comparison with the collected information from the audit trail. This module can store information about known intrusion signatures and/or profiles of normal behaviour. The signatures are most often updated by the security officer, as new intrusion signatures are discovered. This is not an easy task, since the analysis of novel intrusions requires a high level of expertise and this is why signature outsourcing is common. Behaviour profiles, on the other hand, are usually updated at regular intervals, as the users keep using the system, producing new behaviour knowledge.

Active/Processing data - This module stores possible intermediate results such as, for instance, partially fulfilled intrusion signatures.

Alarm - This module handles all the output which, as seen in the diagram, may correspond to an automated response to the intrusion or to an alert for the security officer.

Axelsson's model is useful to describe the general, high level, behaviour of a system of this kind. Regardless, a complete characterization and classification of intrusion detection systems

goes beyond this simple architecture, due to the wide variety of methods and technologies that can be implemented - albeit, in the end, it is still possible to interpret any *IDS* type as a more specific subset of Axelsson's model, which means that it still holds today. Figure 2.2 presents this more complete classification (Pathan, 2014).

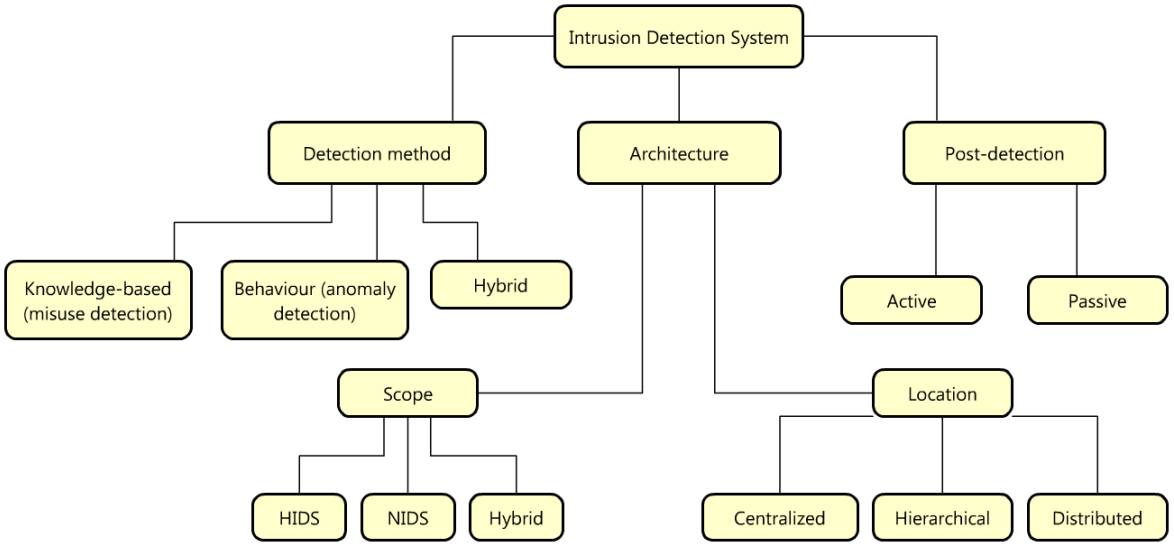


FIGURE 2.2: Characterization and classification of intrusion detection systems.

This classification considers only the *functional* attributes of intrusion detection systems. There is another attribute, a *non functional* one, that should also be considered, which is **usage frequency** (Debar et al., 1999). An *IDS* can be designed to carry either periodic or continuous analysis. Since the former option is possible, sometimes scanners that are used for security assessment are confused with intrusion detection systems. Accordingly, the usage frequency provides a fair distinction between the two.

2.2 Architecture Types

The architecture of intrusion detection systems should be idealized considering three relevant factors: the source of the data to be analysed (the scope), the way tasks are being distributed and the processing components that are built into the *IDS* (together usually referred to as processing location) (Pathan, 2014).

2.2.1 Scope

Considering the source of the information, an *IDS* may be classified as a **host based intrusion detection system**, or **HIDS**, a **network based intrusion detection system**, or **NIDS**, or as a **hybrid intrusion detection system**. Each of these has its own strengths and weaknesses, which will be described next.

2.2.1.1 Host based Intrusion Detection Systems

The host based intrusion detection system collects and analyses the data from a single host. The detection software installed on the host is commonly known as *agent*. It can monitor a wide range of activities, such as the behaviour of applications and processes, changes in the file system, integrity of the system, use of resources, user access and interaction with the system, among many others. Some *HIDS*s can also monitor the data packets that go from and arrive on the host. In fact, some solutions even use dedicated hardware running agent software, and placed in a way that it can monitor the network traffic - technically, these would fit under network based systems, but they work in a similar fashion of a normal host based system, and their monitor activity is more specialized (usually focused on one application, like a web server or a database) than the standard *NIDS* (Scarfone and Mell, 2007).
Agents are developed to monitor servers, hosts, or even applications services. Due to the variety of implementations, de Boer and Pels (de Boer and Pels, 2005) consider four different *HIDS* categories:

1. **Filesystem monitor** - Systems that check the integrity of files and directories.
2. **Logfiles analysers** - Systems that analyse logfiles searching for patterns indicating suspicious activity.
3. **Connection analysers** - Systems that monitor connections attempts to and from the host.
4. **Kernel based** - Systems that search for suspicious behaviour at the kernel level.

The *filesystem monitor* type of agent searches, on a regular basis, for changes in the host's files, comparing to previously stored information, this way detecting if any unauthorized change has occurred. The changes that it searches for are related to file inherent attributes: changes in permissions of the file/directory, inodes, owner and/or group, size, modification and access times, checksums, among others (a filesystem monitor does not necessarily set all of these features). The user is responsible for setting the files and directories that *HIDS* is responsible for monitoring. After creating the initial database, it can be set to perform routine checks on the target, and to present the user with the results. The database needs to be updated regularly to prevent false positives - for instance, the user can legitimately erase a file or directory, which will count as an intrusion to the system. Despite the fact that these *HIDS* types do **not** perform real time analysis, they can still help with intrusion prevention by controlling the access, modification, replacement or deletion of files, this way mitigating, for example, malware installation.

Logfile analysers perform their monitoring and detection tasks through the analysis of logs. This analysis can be done either through pattern matching, pattern matching with correlation between events[1], or anomaly detection (de Boer and Pels, 2005). With pattern matching, the system searches for patterns and compares them to those already known as malicious. The problem with this method is that it looks for individual patterns only - a seemingly innocent line in a log, may not be so innocent if found in more than one log. Pattern matching with correlation between events considers cases like this, effectively providing a more robust protection. On the other hand, an *HIDS* that uses anomaly detection might prove itself even more valuable, depending on the nature of the logs. Indeed, it has no need to know anything about the legitimacy of the logs, requiring only to know what a normal behaviour is.

Connection analysers can stop all the traffic, incoming or outgoing, that contains network, transport or application layer attacks, wireless networking protocol attacks, and unauthorized applications and protocols. This filtering can actually work as a firewall. They exclusively monitor the incoming network connections and, depending on the implementation, these systems can offer features like detection of unauthorized TCP and UDP connections, port binding (which prevents port scans and unauthorized bindings to the given port), host warning and/or blocking, etc.

[1] Analysis carried through pattern matching is often - but not always - a kind of misuse detection.

Lastly, the *kernel based system* refer to one of two system types: either a system that monitors the operating system's kernel or a system that uses kernel methods (in the actual mathematical sense) to detect intrusions. With the first method, it is possible to rely on different data sources: based on system usage, based on the arguments issued to system calls by any process, or the order of system calls themselves, among other intrinsic data sources (de Boer and Pels, 2005). Systems of this kind can be programmed to carry out different types of tasks. There are kernel based *HIDS*s capable of hiding files/directories; killing/protecting processes; detecting local denial of service attacks; preventing someone with root privileges to change a file/directory; and so on. As for the second method, it is directly related to the use of **machine learning** for detecting intrusions. Some of these types of kernels will be approached in the machine learning section.

Scarfone and Mell (Scarfone and Mell, 2007) refer another category worth mentioning - **code analysis**. In this case, agents are capable of using several techniques to identify malicious activity. The methods are: code behaviour analysis, buffer overflow detection, system call monitoring and maintaining application and library lists. With code behaviour analysis, the agent executes the code on a sandbox before running it normally on the host, in search for bad behaviour patterns. This technique can prevent, for example, privilege escalation through a particular piece of code. Buffer overflow detection allows the agent to detect attempts to overflow the stack and/or heap, by paying attention to certain sequences of instructions and attempts to access unauthorized portions of the memory. With system call monitoring, the agent knows which applications and processes are calling which other applications and processes. For instance, with this technique, the agent would recognize a *keylogger* trying to intercept the user's keystrokes. Lastly, by maintaining application and library lists, the agent can monitor those the user or a process attempts to load, and compare this information to lists of authorized or unauthorized applications or libraries.

As it monitors the system, the agent also keeps logging information regarding events it detects over time. These logs are useful later to validate alerts, investigate incidents and to correlate events between the agent and other logging sources (Scarfone and Mell, 2007). The logged data can take many forms: timestamps, event/alert type, prevention action performed (if any), details about the events, etc.

When it comes to true and false positive rates, achieving an acceptable measure of certainty

is challenging, especially when the *HIDS* uses detection and monitoring techniques that have no knowledge of the context under which the detected events occurred (e.g., filesystem monitors and logfile analysers). For example, the host might have rebooted while installing or uninstalling some application, changing system files. These actions are harmless in this case, but they could easily be caused by someone with a malicious intent. In other words, despite the events being accurately detected, their *nature* cannot always be determined without additional context information (Scarfone and Mell, 2007). Given this problem, some products (more commonly those meant for desktops and laptops) prompt the user to supply the context in which the events are occurring (such as upgrading an application). If the user does not respond in a given time limit (minutes or even seconds), the agent acts accordingly, depending on its features and on how it was programmed. Another common way for an *HIDS* to obtain better results is to combine several different techniques, that monitor different characteristics of the host.

Some general advantages of using a *HIDS* are (Bace, 1998, Pathan, 2014):

- Maintaining a strong association between programs and users: identifying the user and the time at which applications or commands were executed;
- Tracking behaviour changes associated with misuse;
- Confronting incoming threats without the need to analyse the whole network traffic;
- Can distribute the work load throughout the available hosts on the network, therefore cutting deployment costs;
- No need for additional hardware.

Of course, there are disadvantages as well (Debar et al., 1999, Pathan, 2014):

- High complexity in management and scalability - every host is configured in its own way, and there is also the need for a constant observation for effectiveness reasons. Some systems have to build profiles of expected behaviour, while others need to be configured with a set of detailed rules that define exactly how applications should behave;
- OS dependent;

- Susceptible to attacks, resulting in corrupted data or data deletion, since it has priority in direct access to the data;

- Vulnerable to alterations to the audit data, if an attack is successful;

- Deep data analysis and very detailed information collection can and will take its toll on system's performance.

2.2.1.2 Network based Intrusion Detection Systems

The network based intrusion detection system collects and analyses data from the network. Note that this does **not** mean that every single bit is examined - off course, this would be too overwhelming.

While monitoring the network for data packets, *NIDS*s usually focus the TCP/IP protocol. Even though most of the analysis is conducted at the application layer, the transport and network layers are also checked, not only in search of attacks at those levels, but also to aid in the analysis of the application layer activity (for instance, an application can be identified through the TCP port number). Some existent *NIDS*s also perform an analysis, albeit limited, on the hardware layer (Scarfone and Mell, 2007).

These systems are normally composed of *sensors*, management servers, consoles and, possibly, one or more databases. The data packets are sniffed out of the network through sensors positioned on "mission critical" spots. Sensors can be of two types (Scarfone and Mell, 2007): **appliance** and **software only**. Appliances are built with specialized hardware and sensor software. The hardware is usually optimized: specialized *NIC*s and *NIC* drivers to capture data packets more efficiently; and specialized processors or other components that can aid in the their correspondent analysis. It is also possible that some part - or even all - of the software resides in the firmware for performance reasons.

The *software only* type has no specific hardware to run on and they are simply installed in common computers. They can either behave like a common application, or include their own OS.

To deploy the sensor, the system administrator has two options, depending on his/her needs. If the goal is to force all the data packets to be analysed before they reach their final destination in order to *prevent* intrusion, the sensor must be placed **inline**, that is, on a place where it can

intercept the data before it reaches the host. This behaviour is similar to that of a firewall. In fact, some of the existing inline sensors are hybrids, providing both the services of *NIDS*s and firewalls. Some of these sensors can also: limit the network bandwidth when a particular protocol is behaving inappropriately; alter malicious content, that is, alter a packet by replacing malicious content with a harmless version (some sensors can even remove infected attachments from emails); reconfigure other security devices present on the network and run third-party programs/scripts to carry additional prevention actions. These sensors are usually placed on segments between networks.

On the other hand, if analysing copies of the network traffic is enough, a **passive** sensor can be deployed. These sensors are often deployed to monitor key network locations, such as a **DMZ**. They can also perform their package capture through a variety of methods (Scarfone and Mell, 2007):

Spanning port - Some network switches have what is called a spanning port, that is able to see all the data traffic. The monitoring is achieved by connecting a sensor to this port. It has the advantage of being a simple and inexpensive method, but it carries some problems too. For instance, due to misconfiguration of the switch, or even to heavy load periods, the port might lack the ability to see all the traffic - in the latter case, the port can even be deactivated for a period of time.

Network tap - With a network tap, the sensor is directly connected to the network medium itself (the actual cable). This tap gives the sensor access to a copy of all the data being carried by the medium. Nonetheless, tapping the network *can* involve network downtime, both in the installation and if something goes wrong afterwards. It can also be an expensive add-on.

***NIDS* load balancer** - The load balancer is a device responsible for aggregating and directing the traffic of a network to the monitoring systems. It can aggregate traffic from different networks, and it distributes the traffic according to a set of predefined rules configured by the system administrator. These rules can, for example, be configured to send all the traffic to more than one sensor or to split the traffic throughout the sensors based on volume (to avoid overwhelming any sensor) and/or based on IP address, protocol or other traits, such as the usual type of data that a given sensor uses to analyse. Splitting traffic

could lead to trouble, however. An example of this would be the case when a threat is composed of two steps, each one analysed by a different sensor. It is feasible that each step, individually, presents no threat at all, which is obviously a problem.

Regarding intrusion prevention, many sensors can attempt to end TCP sessions by resetting them. Unfortunately, this technique often does not work right away, and it is not applicable to non-TCP sessions, such as UDP and ICMP. Even so, and just as inline sensors, some sensors are able to reconfigure other devices in the network and run third-party programs/scripts destined to carry out prevention actions.

Just like *HIDS*s, *NIDS*s are able to access diverse system information. Most of them can at least, for example, identify the hosts in the network (either by IP or MAC address), the hosts' operating systems, the network's overall characteristics (devices, hosts, number of hops that traffic suffers between two devices, etc) and, to a lesser extent, the hosts' used applications. They also have the logging ability in common with *HIDS*s. Nonetheless, given the difference in scope, the type of logged data also differs (there are similar types though, such as timestamps or event information): protocol information, source/destination information, number of bytes transmitted, state-related information, among others.

Although not entirely, the detected events by most *NIDS* are mainly focused on the different network layers (Scarfone and Mell, 2007):

Application layer reconnaissance and attacks - Some of the most usually analysed protocols are DNS, FTP, NFS, POP, SMPT, HTTP (cannot monitor web services though), DHCP and peer-to-peer file sharing software. Common attacks are buffer overflows, password guessing, banner grabbing, malware transmission;

Transport layer reconnaissance and attacks - The most commonly monitored protocols are TCP and UDP. SYN floods, port scanning and unusual packet fragmentation are some of the frequent attacks;

Network layer reconnaissance and attacks - The primarily targeted protocols for analysis are IPV4, ICMP and IGMP. Many systems also consider IPV6. Usual attacks regard spoofed IP addresses and illegal IP header values;

Unexpected application services - These can be detected either through **stateful protocol analysis**[2] methods, that determine the consistency between the activity and the expected protocol, or through anomaly detection, in order to identify changes in network flow and/or open ports on hosts;

Policy violations - *NIDS*s can detect some types of policy violations (e.g., access to inappropriate web sites or use of forbidden application protocols) that can assist administrators on specifying the nature of an unauthorized activity - it can help by revealing TCP or UDP port numbers, IP addresses, visited web site names, and so on.

Throughout history, *NIDS*s have been connected to high rates of false negatives (Scarfone and Mell, 2007). The reason for this is that the earliest technologies relied mainly on signature based detection, which is almost useless against new and/or unknown threats. Due to this fact, these systems nowadays use a combination of misuse and anomaly detection together with stateful protocol analysis, in order to increase the viability and overall performance.
Some of the most prominent advantages of using a *NIDS* are (Pathan, 2014):

- No impact on network performance;
- Attacks can be identified in real time, making it possible for the administrator to respond quickly;
- OS independent;
- Possibly being invisible to attackers, since it does not leave any trace of its presence in the network (opposed to *HIDS* that requires installation on the host, unavoidably leaving traces in the system);
- Requiring simple infrastructures, it has a wide variety of applications.

Nonetheless, the limitations are also prominent (Debar et al., 1999, Pathan, 2014):

- Specific applications might require highly complex protocols;

[2]This detection method was yet to be accounted for as it is generally either not considered as important as the ones referenced in section 2.1 and shown in figure 2.2 - thus ignored -, or considered as a subset of signature based detection (Scarfone and Mell, 2007).

- Cannot detect attacks at a higher level when dealing with encrypted packets (*HIDS*s are useful in these situations: they can be deployed in the network's endpoints to monitor the unencrypted content);

- Restricted applications on segmented networks, especially those with switches;

- Incapable of analysing traffic when the volume of network traffic traces exceeds the system's collecting capacity;

- Need of large recording capacity for data storage (for instance, monitoring the states of TCP connections with large network flows).

2.2.1.3 The Current Trend - Hybrid Solution

As it was being suggested along the two last sections, some problems and limitations that *HIDS*s face can be solved by *NIDS*s and vice-versa. Hence, the idea of using both systems together comes naturally, an this is how *hybrid* intrusion detection systems were born. The main goal is to combine both intrusion detection systems so that they can complement each other, thus maximizing their strengths. As a result, a hybrid system has more flexibility and better performance than any of the systems alone. The hybrid system reunites the best of both worlds: it can act as a *NIDS* by analysing and monitoring the network traffic, while at the same time acting as an *HIDS* by focusing on each host and processing the packets addressed to the system where the *HIDS* component is (Pathan, 2014). The whole system is managed in a **centralized** way (more on this in the next section), no matter in which network segments the sensors are, or which hosts have agents. Figure 2.3 depicts a possible architecture for an hybrid intrusion detection system. The illustrated network represents a LAN connected to the Internet through a firewall. Three different network segments can be distinguished - the local network, the *DMZ* and the Internet. Note that all of these segments have a *NIDS* linked to them. Not only this, but the more critical areas inside the segments also have *HIDS*s attached to them. The *IDS* manager, stationed at the local network, is responsible for the management of all the *IDS*s.

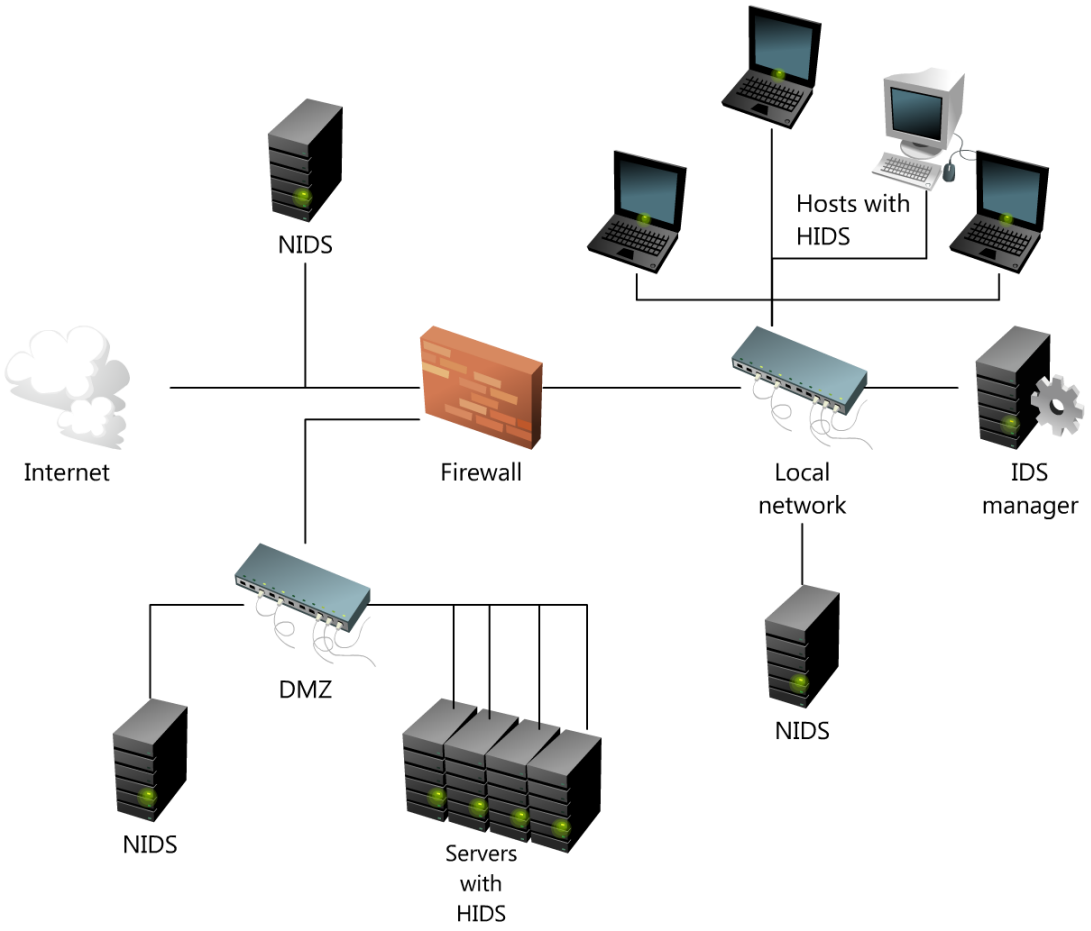


FIGURE 2.3: An example of an hybrid intrusion detection system.

2.2.2 Location

Intrusion detection systems comprise several modules (recall figure 2.1). The *location* of these modules, in conjunction with the system's architecture, allow for the following characterization (Pathan, 2014):

Centralized - The idea of a centralized *IDS* is that there is only one system - the manager or, as it is commonly called, agent-manager - responsible for the event analysis, corelation detection, classification and reaction. The other systems are responsible only for the data collection and transmission to the manager. Figure 2.3 is a possible instance of a centralized *IDS*.

Hierarchical - Hierarchical *IDS*s also have an agent-manager like their centralized counterparts. The difference here is that in a hierarchical system more than one manager can exist, this way establishing an hierarchical organization. For instance, in a given network there might be a manager specifically for all the *NIDS*s, and another manager responsible for all the *HIDS*s.

Distributed - In these systems, there are several manager systems as well, but the components that would normally just collect data can now also process and analyse this data, either fully or partially, before sending it to the manager.

2.3 Reaction After Detection

After detecting a possible intrusion, the *IDS* has to react in some way. There are two possible types of reactions: **active actions** and **passive actions**.

An *IDS* is capable of *active* post-detection actions when it reacts on its own. This type of post-detection is the reason why intrusion detection systems are, nowadays, also called intrusion detection *and prevention* systems (Pathan, 2014). Various examples were already referred on sections 2.2.1.1 and 2.2.1.2.

Passive post-detection actions are the opposite of the active ones. This is the case of *IDS*s that act as decision support systems, merely triggering alarm notifications and sending (or showing) them to the administrator, when a possible attack is detected. Some work has been done on these systems by the **IDWG**, or **Intrusion Detection Exchange Format Working Group**. This group, created through the **IETF**, or **Internet Engineering Task Force** - an organization that thrives to develop and promote Internet standards (NCC, 2012) - sought to define data formats and exchange procedures in order to share information between intrusion detection systems, their modules and management systems through a standard communication language (Feinstein and Matthews, 2007, Wood and Erlinger, 2007).

2.4 Detection Methods

Depending on the chosen methodology for analysing the audit data, *IDS*s can be categorized as **knowledge based** and/or **behaviour based**. Knowledge based intrusion detection systems are commonly referred to as **misuse** or **signature** detection systems, while behaviour based intrusion detection systems are usually known as **anomaly** detection systems. Just like with *NIDS* and *HIDS*, the fusion of these detection methods into a hybrid technique is possible.

2.4.1 Misuse Detection

Misuse detection systems focus on the attack's information (Debar et al., 1999), comparing the collected data to some form of expression of the knowledge previously obtained about the attack. This knowledge is typically expressed through *signatures* or patterns, that correspond to traces of known attacks. A signature can either be built manually or automatically and can take many forms, from a telnet attempt with the username of "root", to an email with a subject of "Free pictures!", along with an attachment called "freepics.exe" (Scarfone and Mell, 2007). Because they compare the analysed activity to the knowledge base directly through pattern matching, misuse detection systems are, consequently, very effective against known attacks. Due to this effectiveness and the scope of their implementation, these systems can potentially keep their *false positive* occurrence at a very low rate. Not only this, but since the attack is known, the performed analysis can produce a high level of detail, making it easier for the security office to engage in preventive or corrective actions (Debar et al., 1999). On the other hand, misuse detection systems are very ineffective at detecting attacks that are unknown such as new threats, disguised threats or even variants of known threats. For instance, on the emailed malware example above, if the attachment was named "freepics2.exe", a signature looking for "freepics.exe" would probably fail to match it (Scarfone and Mell, 2007). For this reason, the signatures database has to be continuously updated with information about new attacks which is a very demanding task (Debar et al., 1999). Ergo, given all this information, one can say that misuse detection systems have good accuracy, but may lack in completeness for their need to constantly update the knowledge base.

This kind of *IDS* is the simplest there is, but its scope is somewhat narrow. These systems

have next to no understanding of network or protocols, and can't track or even understand the state of complex communications (Scarfone and Mell, 2007) - for instance, they have no means to know how to pair a request with the corresponding response, like a request to a Web server and the corresponding response status code of 403 (meaning that the server refused to fill the request). In addition, misuse detection systems are also incapable of keeping track of previous requests when processing the current one. This means that, in a given set of events, if none of them contain a single indication of threat or attack, the *IDS* is unable to detect attacks that compromise more than one event or even the whole set.

Axelsson (Axelsson, 1998), besides signature based intrusion detection, also considers a subset of these systems, which he calls **specification based** intrusion detection. In this case, the knowledge base is filled with information not about attacks or threats, but about *benign* behaviour. The author further merges both signature and specification based intrusion detection into what he calls **policy based** intrusion detection.

Debar (Debar et al., 1999) claims that misuse detection systems might be implemented through one of the following procedures:

1. Expert systems;
2. Signature analysis;
3. Petri nets;
4. State transition analysis.

Expert systems (Jackson, 1990) can represent and reason about a given knowledge base in order to solve problems. The knowledge translates into facts by adding semantic to it (Debar et al., 1999), and is interpreted by the inference engine as a set of *if-then* implication rules - if all the conditions or facts on the left side of a rule (the *if* side) are satisfied, the actions on the right side are carried on. The actions can either trigger other rules or confirm the intrusion. This clear division between control reasoning and formulation of the solution is the main advantage of these systems (Kumar and Spafford, 1994b). However, creating good rules is not an easy task: not only they're subject to human error, but the data from which the control reasoning is devised is either insufficient or *sequenceless*, i.e., there is no default ordering logic to use

when constructing the rules. These knowledge engineering issues have negative impact on the *completeness* of the system (Debar et al., 1999). Other limitation, connected to the system's *performance*, has to do with the use of expert system shells: these require that *all* audit data is imported into the shell as facts before reasoning can take place. **Model based systems** were proposed by Garvey and Lunt (Garvey and Lunt, 1991) as derivatives from expert systems (Kumar and Spafford (Kumar and Spafford, 1994b) classify them as actual variations of misuse detection systems), though armed with additional features. This method bases the detection on the combination of reasoning with attack scenarios *modelled* from behaviour information stored in the database. The system then has an *anticipator* which considers a subset of these scenarios - those most likely to happen - and seeks information on the audit data to either validate or refute them. The information about the next possible behaviour bound to happen is passed on to the *planner*, who then decides how will the hypothesised behaviour show itself in the audit data, translating it to a system dependent audit trail match (Kumar and Spafford, 1994b). As evidence increases for certain scenarios and decreases for others, the active scenarios (models) list is updated. While model based intrusion detection has the advantage of being supported by a well grounded mathematical theory, it may place additional burden on the person that creates the models, as the evidence numbers assigned to each part of the model should be meaningful and accurate.

Signature analysis systems acquire knowledge the same way as expert systems, but process it differently. Instead of converting the audit data to facts, the system keeps a semantic description of the attacks exactly as it can be found in the data. This way, it can create attack scenarios solely from the sequences of events that the attacks generate, or to from patterns of data found on the audit trail (Debar et al., 1999). This version of misuse detection is very efficient, reason for which it is applied for commercial purposes (sections 2.5.3, 2.5.5 and 2.5.6). A *Petri net* (Jensen, 1996) is a famous mathematical modelling language used in a variety of areas such as data analysis, software design, concurrent programming, among others. An extension of Petri nets, called **Coloured** Petri nets (Jensen, 1996), harnesses all the power of the original Petri nets while introducing the power of a high level programming language. With their generality, conceptual simplicity and graphical representability (Debar et al., 1999), the task of writing new signatures is simplified even when the signature has a high level of complexity (although the system administrator should restrain himself from creating overly

complex signatures, as their matching with the audit data can become potentially expensive resource-wise). The *Intrusion Detection In Our Time*, or *IDIOT* project, uses coloured Petri nets. The implementation is explained in (Kumar and Spafford, 1994a).

The *state transition analysis* technique was introduced by Porras (Porras, 1992) on his MSc thesis, and implemented for the first time - on a UNIX environment - by his colleague Ilgun, also in a MSc thesis (Ilgun, 1992). This method regards attacks as sequences of actions caused by the intruder that lead from some initial state (prior to the intrusion) to a compromised state (when the intrusion is complete), where *state* is a snapshot of the system representing the values of all volatile, semi-permanent and permanent memory locations on the system. Between the two states, there has to be at least one *state transition* that the intruder performs to carry out the attack. By identifying the initial and compromised states, the system is capable of also identifying the key actions, also called *signature* actions. Signature actions are those that, if taken out of the attack's execution, would prevent the attack to complete with success. Since this approach only recovers the bare minimum of key actions needed, it simplifies the analysis. Not only that, but it make it easier for the administrator to organize the data using a representation similar to a state diagram.

Kumar and Spafford also refer to *keystroke monitoring* in (Kumar and Spafford, 1994b), which is a method of capturing the user's keystrokes to determine if an attack will occur. They state that this procedure has the disadvantages of the general unavailability of user typed keystrokes and the multitude of different possibilities to represent the exact same attack at a keystroke level, which means that in order for keystroke monitoring to be robust - or even feasible - from a misuse detection's *point of view*, it would need to analyse the semantics of the actual written contents, which clearly raises privacy issues.

2.4.2 Anomaly Detection

An anomaly detection system focuses on information about the system's *behaviour* (Debar et al., 1999). It is based on the premise that security breaches can be detected by monitoring audit data in search of abnormal patterns of system usage, as Denning stated in (Denning, 1987) when presenting one of the first anomaly detection system definitions. The system starts by *learning* the general profile that describes a subject's behaviour when executing some activity

that needs to be monitored, and then collects information over which it can infer the subject's behaviour at a given time, comparing it to the stored profile - in essence, an anomaly detection system searches for deviations from normal behaviour (Axelsson, 1998). Note that "subject" is being used as a generic way of addressing the audited entity - it can be a user, a process, a network, a set of parameters, etc. If the audited behaviour's deviation from the learned profile surpasses a certain threshold or is simply large enough, the activity is considered a possible intrusion. Although the concept is fairly straightforward, the actual division between anomalies and normal data is quite challenging to achieve. The reasons for this may vary (Chandola et al., 2009):

- Defining a normal region is not a trivial task. If, for instance, there are anomalous examples close to normal data points, the definition of the normal region might be ambiguous, treating anomalous observations as normal and vice-versa;
- When anomaly detection systems are used to detect malicious behaviour, the attackers can possibly adapt their intrusion in order to make it appear as normal, thus hindering the definition of normal behaviour;
- In domains where behaviour is expected to evolve through time, whatever represents normal behaviour in the present might not be sufficient in the future;
- The exact definition of an anomaly depends on the domain where it is applied: a small deviation from normal in a medical domain (for instance, a slight fluctuation in body temperature) might be regarded as an anomaly, while the same deviation in a stock market domain (for example, variation on the value of a stock) is probably considered as normal. For this reason, a technique developed for a domain cannot be applied to another domain in a carefree fashion;
- Availability of labelled data for training and/or validating anomaly detection models is usually a major issue: there is very little chance that a company's data about the behaviour of some entity will be organized, let alone correctly labelled. As such, the developer will probably have to take a *leap of faith* and assume that all the data represents normal behaviour (it actually is relatively safe to make this assumption, as long as the

majority of the examples are normal - the detection will still work fairly well even with some anomalies amidst normal observations);

- Apart from anomalies concealed within normal regions, real world data tends to be noisy, which sometimes can make normal examples dangerously similar to anomalies, thereby blurring the distinctiveness between the two types of data.

With the above information in mind, one can state that anomaly detection systems *can* be considered *complete*, albeit rather problematic when it comes to *accuracy* (Debar et al., 1999). Regardless, in cases when the system flags acceptable behaviour as suspicious, the administrator can, at any time (if the system allows it), mark it as acceptable, so that from then onwards that same behaviour pattern is treated as such (Pfleeger and Pfleeger, 2003).
As it would be expected, the performance of an *ADS* is directly influenced by external factors such as the type of input data and the type of anomalies the system has to deal with. In fact each case is unique, and the behaviour profile creation algorithm has to be carefully planned and subsequently tweaked in accordance with these external influences.

2.4.2.1 Input Data

The input of an *ADS* is in general a set of data instances, often called *dataset*. Each example of the set (also commonly referred to as *observation*, *object*, *record*, *sample*, *point*, *vector*, among others) consists on a set of *attributes* (also known as *features*, *variables*, *dimensions*, *fields*) which define the sample in question. These attributes are generally seen in one of three types (more types, as well as other conventions, exist):

1. **Binary** - Only 0 or 1 as possible values;
2. **Numeric** - Quantitative type. Describes a measurable quantity as a number, which can either be an integer or a real value;
3. **Categorical** - Qualitative type. Describes a quality or characteristic of a data point. These features should be mutually exclusive - an observation cannot have two features of the same category - and tend to be represented by non-numeric values, although there

are cases where numbers are used. They can also be further divided into two groups: *ordinal* variables, which can be logically ordered in some way (eg. A, B, C) and *nominal* variables which have no underlying ordering criteria.

A data point can have one or more dimensions. In the case of multivariate examples, the dimensions can be of different types. It is the nature of these types that defines which algorithmic techniques can be used to the anomaly detection. For instance, a method that builds a profile model through numeric operations cannot be efficiently applied to categorical data represented by non-numeric values. Although, it is possible to alter the raw data into another type without losing information. In fact, categorical values are almost always translated into a more basic type, such as numeric or binary, or even broke down into more simple features (the anomaly detection techniques used, in spite of being efficient, may (and probably will) have trouble interpreting complex categorical data, such as IP addresses or product names).
Data points can be related to one another. The more common manifestations are *sequence data*, *spatial data* and *graph data* (Chandola et al., 2009). Sequence data is the case where the examples are somehow sequentially ordered, e.g., time-series, genome or protein sequences. Spatial data relates data points to the neighbouring instances. This is the case of vehicular traffic and ecological data, for instance. When this type of data has a temporal (sequential) component, it becomes *spatio-temporal data*, which is the case of climate data for example. Lastly, in graph data the examples are represented as vertices in a graph, connected to other vertices with edges. These relations become more relevant when they are taken into account together with anomaly types.
Finally, the dataset can be either **labelled** or **unlabelled**. Data is labelled when each data instance has a special attribute that specifies the kind of example - normal or anomalous. The labels are added by a human expert which might require a considerable effort specially for labelling anomalous data: while normal behaviour is limited to the actual data, anomalous behaviour, often being dynamic in nature, can have a variety of unknown forms. The labelling will also influence the kind of methods that one can apply to the problem. *Supervised anomaly detection* is the name given to the techniques that use datasets with both normal and anomalous examples labelled. The idea is to build a model able to distinguish between normal and anomalous classes, and any unseen observation is classified using the model to define the class in which it belongs, with some degree of assurance. However, there are some

problems with these methods. Firstly, it is extremely difficult to have enough anomalous data to accurately represent the anomalous class, as anomalies can take many different forms and it is impossible to predict them all. Apart from this, the examples available tend to be far fewer than the normal cases - this will cause the classes to be skewed, which is something that one should avoid at all costs since it will not allow for a trustworthy detection. Secondly, obtaining accurate examples of anomalies is remarkably challenging in most cases. In fact, there are cases where anomalous examples are impossible to recover unless artificial methods are used, which of course are not optimal. Some of these methods were proposed in (Theiler and Cai, 2003), (Steinwart et al., 2005) and (Abe et al., 2006). The methods that use only one class are called *semi-supervised anomaly detection*, and the used class is generally the one representing the normal behaviour, for the same reasons explained above. Since these techniques do not have to deal with the same problems as their fully supervised counterparts, they are more widely applicable (Chandola et al., 2009). The process aims to build a model based on normal behaviour, and any new data point that does not fit into the model is considered an anomaly. There are methods that consider the existence of only the anomalous class, such as in (Dasgupta and Nino, 2000), but these are rarely used, given the problems already stated. The last method type is called *unsupervised anomaly detection*, and as the name suggests, the techniques of this kind do not use labelled data. These techniques do not require training, and rely solely on the assumption that, in the test set, the majority of the observations represent normal behaviour - of course, if this assumption does not hold, a high false alarm rate can be expected. Semi-supervised methods can be adapted to perform in an unsupervised environment, by sampling a subset of the unlabelled data as training dataset. Of course, this is only reliable if the same assumption referred above holds (Chandola et al., 2009).

These different methods can be set in a **static** or **online** detection environment. On static anomaly detection, the model is created from a given dataset and remains as it was created until a new model is needed. Although simpler, this method fails to adapt to changes in the target's behaviour. In turn, online anomaly detection is more complex to implement, but can adapt to changes in behaviour. The disadvantage is that, by slight changes in behaviour over time, the *ADS* can be tricked into accepting anomalies as cases of normal activity.

2.4.2.2 Anomaly Types

Anomalies are data patterns that deviate from the patterns considered as representatives of normal behaviour. Figure 2.4 depicts an ideal situation, since the data is explicitly separated (Chandola et al., 2009). The two normal regions, N_1 and N_2, are where most of the data examples lie. Observations that are acceptably far away from these regions - points o_1, o_2 and region O_3 - are anomalies.

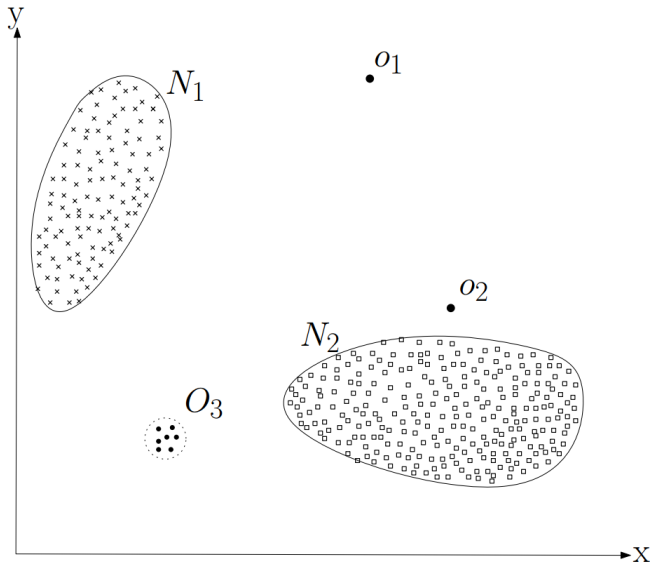


FIGURE 2.4: Anomaly example in a 2D dataset.

According to (Chandola et al., 2009), anomalies can be classified into three different categories:

1. **Point Anomalies** - If a single data instance can be considered anomalous regarding the rest of the data, then it is considered an anomaly point. Points o_1 and o_2 on figure 2.4 are such examples. A simple real life case would be tracking the amount of money one spends with a credit card. If, on one transaction, the value was way higher compared to normal activity, the event would be tagged as an anomaly;

2. **Contextual Anomalies** - If an observation is considered an anomaly only when a specific context applies, then the data is defined as a set of contextual or *conditional* anomalies. The context has to be specified as a part of the problem formulations and is defined by the attributes of the dataset, which can be one of two types: *contextual attributes*,

that determine the context or neighbourhood for the observation (for example, in a spatial dataset, longitude and latitude are contextual attributes) and *behavioural attributes*, which define the non-contextual features of the observation (for example, in a spatial dataset describing the average rainfall around the world, the amount of rainfall in a given location would be a behavioural attribute). These anomalies are mostly considered in spatial and time-series data. Again, using the credit card example, the time of purchase could be a contextual attribute: if a user always spends 100€ every month and 1000€ in Christmas, spending 1000€ in August would be considered an anomaly.

3. **Collective Anomalies** - Collective anomalies are the cases where a group of related data points are compared to the dataset and turn out to be anomalous, even if the individual examples are considered normal. This type of anomaly can only occur if the observations have some type of relation between them. Once more, using the credit card example: the amount of money spent per transaction could be a feature that, in a collective anomaly perspective, forms a new feature called *amount of money spent per day*. If the user usually spends 20€ per day, and suddenly spends 50€, this event will be considered anomalous.

2.4.2.3 Profile and Detection

To create the base profile, the system has to monitor the characteristics (or behaviour) of a target activity over a certain period of time (Scarfone and Mell, 2007). A profile might show for instance that during a typical workday, 15% of a user's emails have files attached to them. The *ADS* can then choose among a set of well defined methods (sections 2.4.2.4, 2.4.2.5, 2.4.2.6 and 2.4.2.7) to compare current behaviour to this previously defined profile, giving it the ability to detect anomalies such as when the user starts attaching files to every email, which can possibly even represent a data leak problem or insider attack.
In the last example, the kind of malicious activities that it could instantiate - consuming the host's resources, sending a large number of files or emails through the network, as well as starting new connections continuously - would easily trigger the *ADS* responsible for monitoring the host (Scarfone and Mell, 2007). In (Kumar and Spafford, 1994a), Kumar and Spafford divide alarms into four different possible types, each one with non zero probability:

1. **Intrusive but not anomalous** - When the *ADS* fails to detect an intrusion, classifying it as normal behaviour. These situations are usually known as false negatives;

2. **Intrusive and anomalous** - Ideal situation when the system performs as expected, correctly detecting malicious activity. These are often called true positives;

3. **Not intrusive but anomalous** - False positive situation, when the user does not mean any harm (no intrusive behaviour), but the system still considers his/her activity as anomalous (very common alarm in misuse detection, due to the nature of the algorithm);

4. **Not intrusive and not anomalous** - The other ideal situation, where the system does not raise any issues regarding a benign user's activity. These situations are commonly referred to as true negatives.

Note that this particular description considers that the *ADS* flags anomalies as *positive* values and normal behaviour as *negative* values. Depending on the implemented detection method, the opposite can also occur. For the rest of this section, the above case will be assumed for coherence.

Figure 2.5 shows a **probability distribution function** (or *PDF*) representation of *ADS* alarms. Both behaviour types follow a normal distribution. The threshold line represents the decision boundary between what the algorithm considers normal and intrusive behaviour. The best case scenario would be having the two distribution functions completely separated, which unfortunately is theoretically unachievable. Instead, the goal is generally to attempt to minimize both false negatives and false positives. These rates are both deeply connected to the threshold value, which further hardens the task of minimizing them: as seen in the figure, if the threshold is set at a low value, the number of false positives will be lower, but the number of false negatives will increase. On the other hand, if the threshold value is set too high, the number of false negatives drops, as the number of false positives rises. As a result, the trade-off between the rates has to be handled with care, also taking into consideration the use that the *ADS* is meant for (Jain et al., 2004): the more critical and sensitive information the system where it is installed handles, the higher the threshold should be, thus avoiding the more critical false negatives.

Anomaly detection systems, since their conception, go through two distinct phases before they

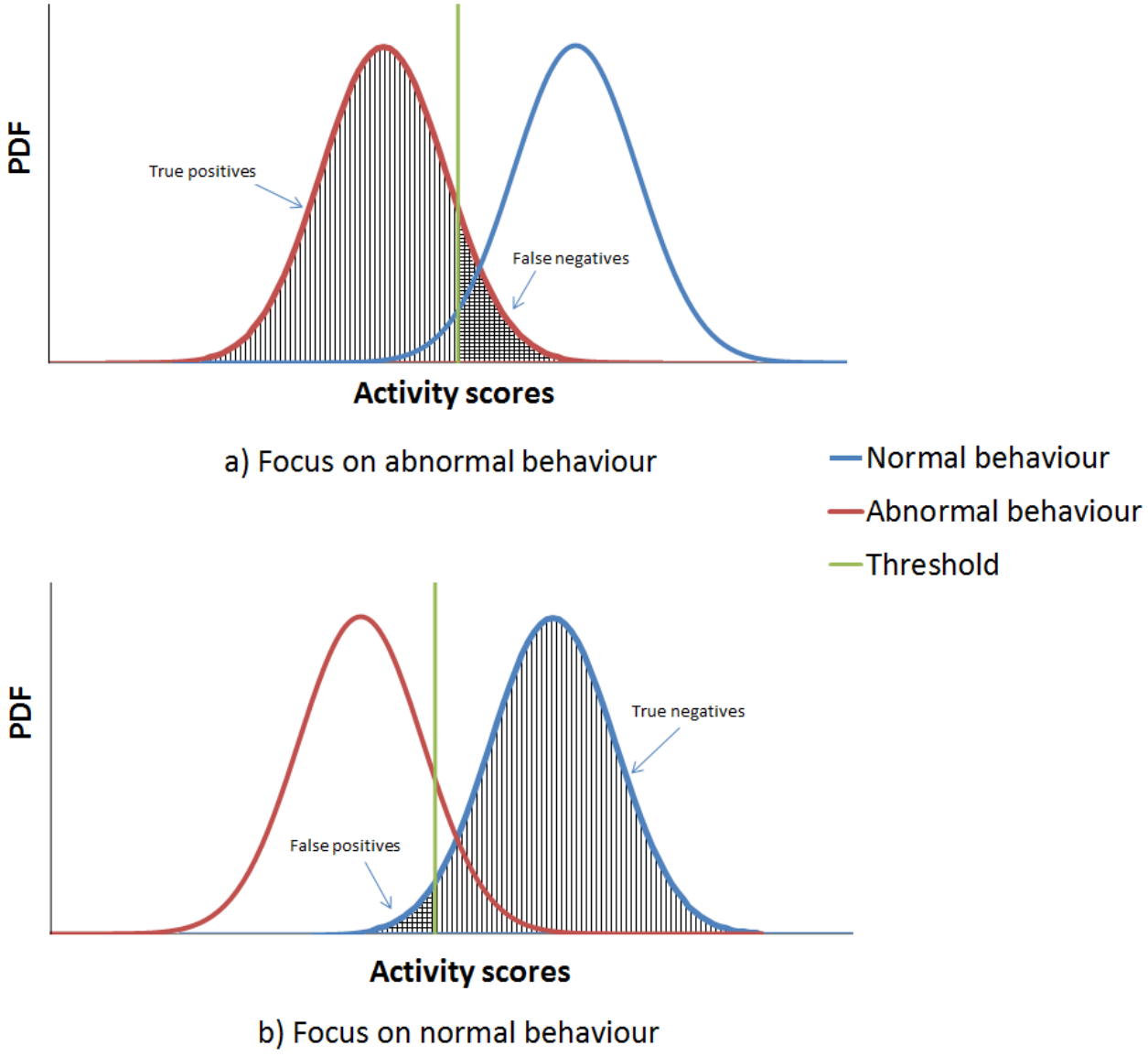


FIGURE 2.5: Alarm situations defined by probability distributions.

are deployed (Patcha and Park, 2007). First comes the *training* phase which was already summarily described: it is where the *ADS* creates a model of the subject's behaviour. The dataset used has to contain enough information (of normal behaviour - unlabelled - or both normal and anomalous behaviour - labelled -, depending on the method) for the algorithm to build the behaviour profile - the more information there is, the more accurate the profile will be, but the training will consequently take more time. Axelsson describes two different types of anomaly systems, regarding the training method (Axelsson, 2000):

Self-learning systems - As the name suggests, a system of this type can start monitoring

and analysing information and events by itself, and then attempt to build the behaviour profile through what it observed;

Programmed systems - In this approach, the system needs the administrator to aid in the construction of the behaviour profile, by manually inputting the information *he/she* thinks is relevant.

The generated model can either be *static* or *dynamic* (Scarfone and Mell, 2007). A static model remains unchanged until the *ADS* is explicitly ordered to generate a new one. Dynamic models however, adjust over time, as events are observed. They have the advantage to better adapt to the ever changing system and network configurations, while a static version will become deprecated in a short amount of time.

When the training phase is over, it yields to the *testing phase*. Here the resulting model from the previous phase is tested. The dataset used is composed by (labelled or unlabelled) information either solely about normal behaviour or both normal and intrusive behaviour, so that careful and precise tweaking of the detection method is possible. This phase is essential to calibrate the *ADS*, even if in the cases when anomalous information is used, all of the intrusion possibilities are not covered (which is indeed an impossible level of detail to achieve). After testing and calibrating, the *ADS* is ready to be deployed. When a new instance appears, it will be processed by a classifier algorithm using the model as reference. The output, depending on the implementation, will be either a **score** or a **label**. The score, or anomaly score, can be compared to a predefined threshold, and the result of this comparison defines if the observation is normal or anomalous. In this case, multiple levels (multiple thresholds) can be defined, thus ranking anomalies by score which can translate to the seriousness of the threat. On the other hand, a label simply tells immediately if the example was considered normal or anomalous.

Besides the high possible rate of false alarms - along with the difficulty of choosing a threshold value - and the possible large amount of time that it might take to train the algorithm, anomaly detection systems are far more complex to implement than misuse detection systems. Different architectures and methods have been proposed for anomaly detection over the years. The most common methods to implement an *ADS* are statistical methods (Javitz and Valdes, 1991, Manikopoulos and Papavassiliou, 2002) and data mining methods (Lee and Stolfo, 2000, Portnoy et al., 2001), the latter usually translating into classification (Lane and Brodley, 1997,

Shon and Moon, 2007)[3] or clustering, although other methods were still developed or tested - expert systems (Denning, 1987, Dowell and Ramstedt, 1990, Vaccaro and Liepins, 1989), *computer immunology* (Forrest et al., 1997), *user intention identification* (Spirakis et al., 1994, Spyrou and Darzentas, 1996), and a few other approaches (combinations of different methods are also possible (Javitz and Valdes, 1991)).

2.4.2.4 Statistic and Probabilistic Methods

Statistical methods for anomaly detection have their foundation on a single key assumption (Chandola et al., 2009): normal data instances occur in high probability regions of a stochastic model, while anomalies occur in the low probability regions. These techniques fit a statistical model to the data - most commonly for the normal behaviour -, so that they can afterwards infer if an unseen behaviour instance belongs to this model or not. If this instance has a low probability of being generated by this model, it is classified as an anomaly. The model can be fit either through parametric or non-parametric methods, the only difference being that parametric techniques use a fixed number of parameters as they know beforehand the nature of the distribution, whereas non-parametric methods have no such knowledge, thus increasing the number of parameters as the quantity of audit data available increases.
With parametric techniques, the anomaly score of an observation x is the inverse of the probability density function, $f(x, \Theta)$, Θ being the set of parameters estimated from the audit data (Chandola et al., 2009). Another option is to use a *statistical hypothesis test*, where the *null* hypothesis H_0 states that the observation or instance x has been generated through the estimated distribution. Hence, x is considered an anomaly if the statistical test refutes H_0. Parametric methods can be classified with regard to the nature of the distribution:

Gaussian model based - Methods of this type expect that the data is generated from a Gaussian distribution. The parameters are estimated using *Maximum likelihood Estimates*, or *MLE*, and the distance of the instance to the estimated mean is its corresponding anomaly score (Chandola et al., 2009). To then decide if the observation is an anomaly or not, its

[3]Note that the previous references are directed to some implementation methods, and not the most known implementations themselves.

score is compared against a threshold. Different techniques will use different thresholds, and different measures to obtain the anomaly score.

Regression model based - Anomaly detection techniques based on regression models are divided into two steps. Firstly, a regression model fits the data. Afterwards, for each test instance, the *residual* for the instance is used to compute the anomaly score (Chandola et al., 2009). This residual is the portion of the instance that cannot be explained by the regression model, and its magnitude can actually be used as the anomaly score (though other measures also exist). Problems might arise if the training data has anomalies, as the generated parameters can be influenced, thus rendering the regression model inaccurate. Fortunately there are techniques to deal with anomalies on training data, such as the popular *robust regression* (Rousseeuw and Leroy, 1987). Regression is often also considered as a classification method.

Mixture of parametric distributions - Techniques of this nature can be categorized into two subsets. The first one is comprised by techniques that model normal observations and anomalies as distinct parametric distributions (Chandola et al., 2009). The testing phase for this subset has to determine which distribution - normal or abnormal - the test instance belongs to. Abraham and Box, in (Abraham and Box, 1979), assume that both normal and abnormal data sets follow a Gaussian distribution, the latter having a larger variance - $N(\mu, \sigma^2)$ and $N(\mu, k^2\sigma^2)$, respectively. The test observations are tested using the *Grubbs'* test, which is used to detect anomalies in a univariate dataset (Grubbs, 1969) (also called *maximum normed residual test*), and flagged accordingly to the result. Other methods were also proposed, for instance, in (Abraham and Box, 1979, Eskin, 2000).
The second subset's techniques model only the normal observations as a mixture of parametric distributions (Chandola et al., 2009). A test instance is flagged as an anomaly if it does not belong to any of the models. These techniques have mostly used Gaussian mixture models (Agarwal, 2007).

Finally, non-parametric methods. With these techniques, the model is not initially known, but instead derived from the data. They also make fewer assumptions regarding the data, when compared to parametric techniques (Chandola et al., 2009). Some of these techniques are as follows:

Histogram based - With this technique, histograms are used to maintain the normal data profile. Techniques of this category are often referred to as *frequency* or *counting* based (Chandola et al., 2009), and require normal data to build (Javitz and Valdes, 1991) - if labelled anomalous data is available, anomaly histograms can be constructed as well (Dasgupta and Nino, 2000).

A basic implementation for univariate data is defined in two steps. The first step is building an histogram based on the values corresponding to a given feature on the training data. In the next step, the test instances are tested by the algorithm, in order to see if they fall into any of the histogram's bins. If they do, they are considered normal; if they do not, they are flagged as anomalous (Chandola et al., 2009). A variant of this technique implies the association of an anomaly score to each test instance, regarding the frequency of the bin in which it falls. The size of the bin is also relevant: if the bins are small, a large number of test observations will fall into empty or low bins, resulting in a high false alarm rate. If, however, the bins are large, anomalous instances will fall into frequent bins, thus raising the number of false negatives. Consequently, optimizing the size of the bins is a main challenge of this technique (Chandola et al., 2009).

The basic technique for multivariate data is to construct *attribute-wise* histograms. This means that, for each test observation, the anomaly score for each of its attributes is calculated as the height of the histogram that contains that attribute. The *per-attribute* scores are then aggregated to obtain the final, general score. A variant of this method is applied in (Mahoney and Chan, 2002).

Kernel function based - Methods based on kernel functions are similar to parametric methods described before. The main difference is the density function estimation technique used (Chandola et al., 2009). A possible technique of sorts is the *parzen windows estimation* (Parzen, 1962). For example, Desforges (Desforges et al., 1998) proposed a method to "identify abnormal or unexpected conditions from measured response data", where they estimate the *pdf* function through a kernel method.

Below are some advantages in using statistical methods (Chandola et al., 2009):

- If the assumptions about the data distribution hold true, the method is then a statistically justifiable solution for anomaly detection;

- The anomaly score obtained is related to a confidence interval, which can be further used as additional information when making a decision for any test observation;

- If the distribution estimate phase stands firm against anomalies in data, then it can operate under an unsupervised setting with no need for labelled training data.

And some considerable problems (Chandola et al., 2009):

- Statistical methods rely heavily on the assumption that data is generated from a particular distribution. This assumption is often false, especially when dealing with highly dimensional datasets;

- Choosing the best statistic to implement is not always a straightforward task (Motulsky, 2014);

- Histogram based techniques, though fairly simple to implement, fail to capture the interactions between different attributes, when applied to multivariate data. An anomaly can potentially have attribute values that are quite common individually, while their combination is extremely rare.

2.4.2.5 Data Mining

Intrusion detection can be seen from a data-centric point of view as a data analysis process (Lee and Stolfo, 2000). Data analysis is not a trivial task to be done manually. It becomes even harder and rather inefficient in cases where the most important patterns that can be found are not visible to the naked eye. Given this, when confronted with data to be analysed researchers normally resort to **data mining**. Data mining is a subset of computer science that aims to uncover hidden patterns from typically large and multivariate datasets. Often, authors blend data mining with classification. As an example, Patcha and Park describe **neural networks** and **support vector machines** in (Patcha and Park, 2007), which are typically associated to **machine learning** (a subset of classification), as data mining methods. The concepts are in fact similar, which creates the need to distinguish them clearly. Data mining techniques are executed by a human in order to find, as stated before, *unknown* patterns in data. Machine

learning, on the other hand, can be carried without direct human interaction, and its goal is to *predict* behaviour based on *known* patterns that it learns in the training phase. Even knowing the definition of each concept, it still is troublesome to distinguish them, as they tend to overlap each other: data mining uses classification techniques, in spite of the different goal; classification uses data mining methods, either as a preprocessing step (to improve the accuracy of the learning phase) or as unsupervised learning.

Data mining can also be understood as a superset of all these similar technologies - statistics, machine learning, pattern recognition and matching, etc - as they all are used to find some type of information in a database or dataset (John, 1997). With time, it became a business intelligence *buzzword* with a life of its own[4].

Data mining is the data analysis step in **KDD**, or **knowledge discovery in databases** (Fayyad et al., 1996a). *KDD* corresponds to the whole process from data gathering to the output of meaningful data patterns. To better understand data mining, one should understand the *KDD* process as a whole. This process is divided into several steps that may vary in number between publications but, in the end, the definition of the whole process holds for every case. The steps here described follow the line of thought in (Fayyad et al., 1996b), and are illustrated in figure 2.6 (although there are boxes representing more than one step given their deep connection).

The first thing to do is to completely **understand the application domain**. One has to understand the goal of the application and everything relevant to the achievement of that goal. An incomplete knowledge of the domain might force one to go back to this first step when he or she is already writing code. After this step, it is time for the **data selection**. The developer has to dig deep into all the data of the application domain, and uncover all the relevant information to construct the initial dataset. When the initial data is selected and the raw dataset is ready, it still has to be **preprocessed** and **cleaned**. In these two phases, the idea is to clear the dataset from noise, outliers - or at least account for them -, define strategies to handle missing values and/or *DBMS* related issues such as data types or data redundancies. As this point, the data is ready to the next stage, which regards its **reduction** and/or **projection**. The goal is to select and extract useful features that are representative and informative about

[4]The book "Data Mining: Practical Machine Learning Tools and Techniques"(Witten et al., 2011) was once called "Practical Machine Learning" - "data mining" was introduced later for marketing purposes (Bouckaert et al., 2010).

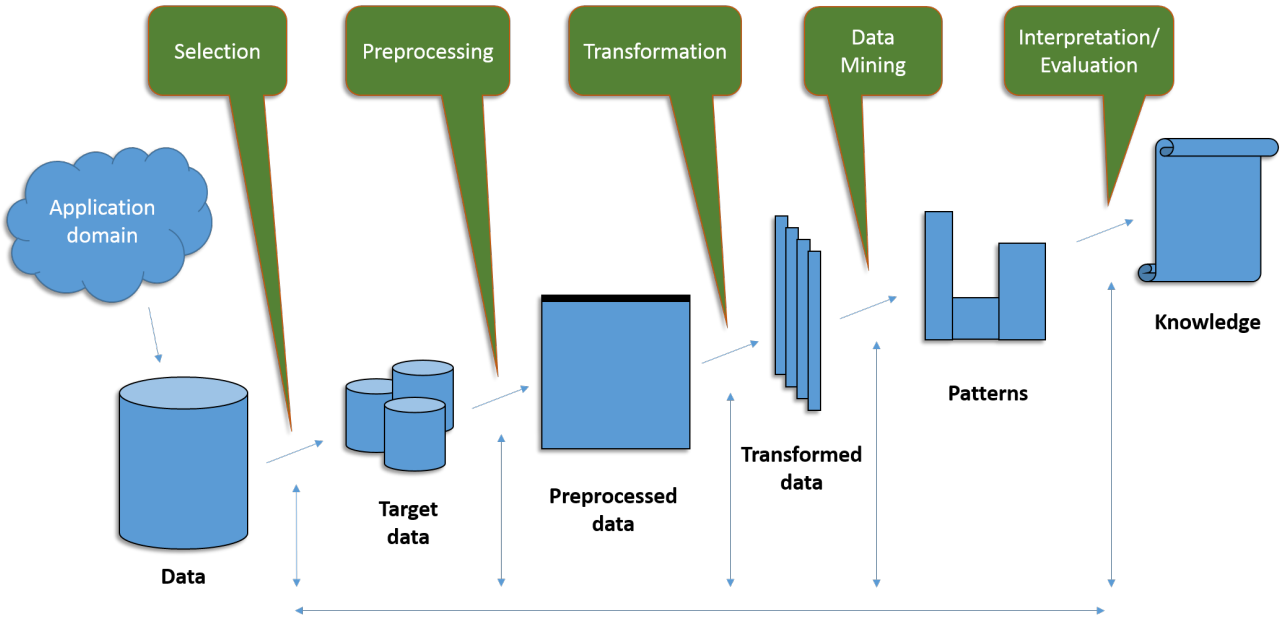


FIGURE 2.6: *KDD* process breakdown.

the data, as well as using *dimensionality reduction* or other transformation methods to reduce the number of actual variables in the dataset. One of the most well known dimensionality reduction algorithm is called **Principal Component Analysis**, or **PCA**. This statistical procedure is capable of two different, yet related, goals: to allow for a visualisation of the data when searching for more obvious patterns (particularly useful with high dimensional datasets) and to dim the effects of the *curse of dimensionality*. This means that for a dataset consisting of a large number of correlated variables, *PCA* will try to reduce the number of variables while trying to maintain as much as possible of the variance present in the dataset (Jolliffe, 2002). The reduction is achieved by transforming the old, **correlated** variables into new **uncorrelated** variables, which are then ordered by descending order in variance level, so that the first few retain most of the variance present in all of the original dataset. The new values are linear combinations of the old values (Jolliffe, 2002). After running *PCA*, it may be possible to still reduce even further the dimensionality of the dataset, as the first few new variables - called *principal components* - might hold most of the variance of the original dataset, which allows for the rest of the variables to simply be ignored with minimal loss. The plotting of the principal components leads to the visualisation objective: when plotting them against each other, the importance of certain features over others as well as certain patterns might be uncovered.

The next stage of the *KDD* process is, finally, **data mining**. In (Fayyad et al., 1996b) this

stage is divided into three different steps. The first step is to choose the **purpose** of the outputted model by the data mining algorithm - whether it should classify, summarize or cluster, for instance. The second step involves choosing the actual **data mining algorithm**. This step is highly dependant of the type of data available, and should conform to the purpose of the *KDD* process - for instance, the developer might be more interested in understanding the model, rather than in its predictive power. The third and final step is the actual data mining, i.e., creating a model from the dataset that is capable of uncover important patterns in a particular form depending on the chosen algorithm. After the data mining stage, come the two final steps: the **interpretation** of the data, which consists on analysing the uncovered patterns, inferring their usability and relevance (there might be the case where patterns are irrelevant or redundant) and translating the useful ones to a form that humans can understand, and the **usage** of the newly uncovered knowledge, which can mean different things, such as incorporating the knowledge into a system that can act according to it, document and report the knowledge to any interested parties, or comparing the knowledge to its previously believed or extracted counterpart. An important aspect of the *KDD* process is that, at any step, one can move back to an already cleared step if needed.
As it was already stated, data mining for anomaly detection is commonly implemented through classification and clustering. These methods are clarified in the next sections.

2.4.2.6 Classification

Classification has three most commonly used formats: supervised, semi-supervised and unsupervised learning, which were already referred in section 2.4.2.1. These procedures are implemented under the assumption that a classifier who can distinguish between normal and anomalous data can be learnt in the feature space (Chandola et al., 2009). The built model can be one of two types: *one-class* and *multi-class*. One-class classification based anomaly detection methods assume that the training dataset has only one class label. These procedures learn how to implement a discriminative boundary around the normal instances, treating everything out of that boundary as anomalous. On the other hand, multi-class methods are used when the training dataset has labelled instances belonging to more than one normal class - even so, there can exist anomalous cases that do not belong to any of the classes. For each

class, there is a classifier. Some variants of this method associate a confidence level with the prediction made. Figure 2.7 depicts an example for each classification type (Chandola et al., 2009).

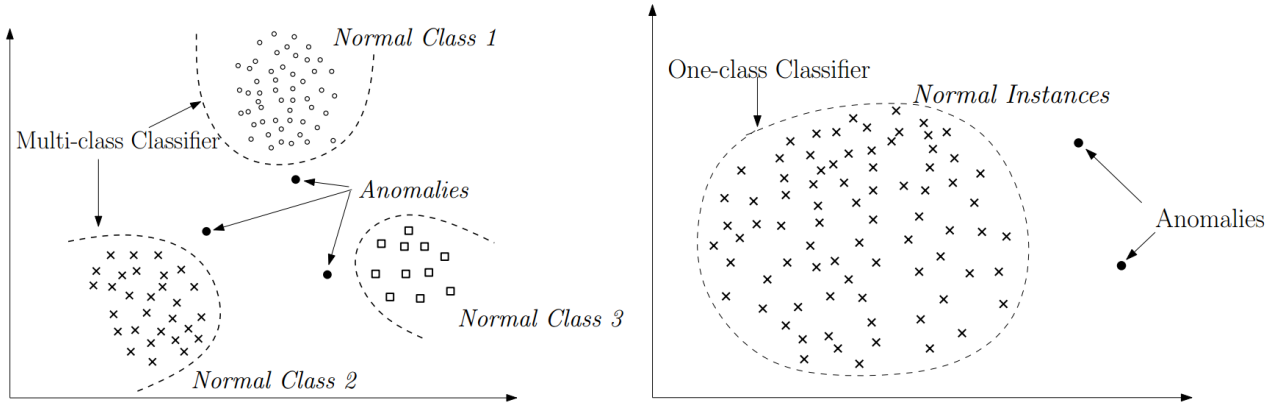


FIGURE 2.7: Classification procedures.

A typical classification algorithm estimates a decision function $f : \mathbf{R}^N \mapsto \{\pm 1\}$ for the training data, that is, a dataset $\mathbf{D}$ of l observation vectors $\vec{x_i}$ with n dimensions and class labels y_i, such that f will correctly classify new examples $(\vec{x}, y)$, i.e., $f(\vec{x}) = y$. This function has to be carefully selected since, if no restriction is imposed, it is possible to obtain a function that outstandingly classifies training data in a correct way, but fails with test and unseen data. This is known as **overfitting** and is a common mistake. Regardless, cases where the function is just too simple to properly fit the data, not even showing acceptable performance with the training set, also occur, and this is called **underfitting**. To avoid these two extremes, one should search for a function in between the two cases that **generalizes** for different datasets. As of today, there is one off the shelf classification method that has been the centre of attention for both its relatively simple implementation and, most importantly, its effectiveness. This method is called *Support Vector Machine* classification, or **SVM**, and it was originally designed for binary (two-class) classification only (Boser et al., 1992, Cortes and Vapnik, 1995), in a supervised learning environment. Over the years, new formulations arrived, and today *SVM* can be used for classification, regression, feature extraction, distribution estimation, among other similar operations, in supervised, semi-supervised or unsupervised learning environments.

In the original *SVM* concept (all the others branch from it), the *SV* classifier will be a *hyperplane* defined as

$$\vec{w}^T \cdot \vec{x} + b = 0, w \in \mathbf{R}^N, \quad b \in \mathbf{R}, \tag{2.1}$$

which corresponds to the decision function

$$f(\vec{x}) = \text{sign}(\vec{w}^T \cdot \vec{x} + b), \tag{2.2}$$

where $\vec{w}$ is a *weight vector* normal to the hyperplane and the independent term b is known as the *bias*, and it translates the hyperplane away from the origin. An infinite number of hyperplanes exists, but the optimal hyperplane is the one that establishes the largest possible *margin* between the closest data points to it - the so called **support vectors**. These observations completely define the optimal hyperplane, so if they are corrupted by noise or not informative, the algorithm will not do well. Figure 2.8 illustrates a simple example where a hyperplane (the line in the middle) separates red balls from blue balls with the largest margin. The optimal

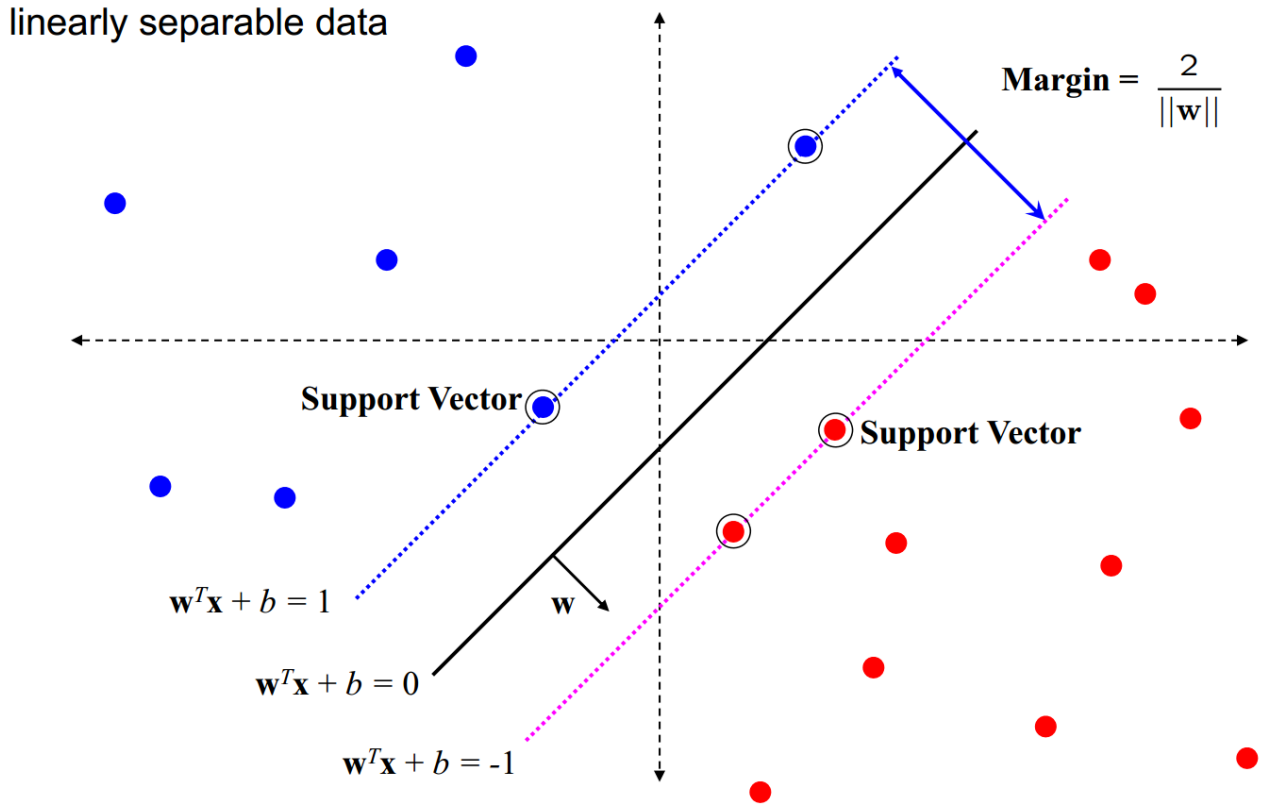


FIGURE 2.8: Graphical example of linear classification with the SVM algorithm.

hyperplane is perpendicular to the shortest line connecting both convex hulls (dotted boxes enveloping the classes), intersecting its midpoint (hence, at the same distance from both sides). This shortest line corresponds to the distance between the parallel hyperplanes $\vec{w}^T \cdot \vec{x} + b = -1$ and $\vec{w}^T \cdot \vec{x} + b = +1$, i.e., the size of the margin which, through these hyperplanes, can be mathematically reduced to $\frac{2}{\|\vec{w}\|}$, as shown in the figure. Having said that, since the algorithm wants to maximise the margin, it will in reality try to minimise $\|\vec{w}\|$. In addition, constraints are needed in order to assure that the data points are correctly classified. In a more formal perspective,

$$\begin{cases} \vec{w}^T \cdot \vec{x}_i + b \leq -1 & \text{if } y_i = -1 \\ \vec{w}^T \cdot \vec{x}_i + b \geq +1 & \text{if } y_i = +1 \end{cases} \tag{2.3}$$

which can be rewritten as simply

$$y_i \cdot (\vec{w}^T \cdot \vec{x_i} + b) \geq 1. \tag{2.4}$$

By joining equations 2.2 and 2.3 with the minimisation of $\|\vec{w}\|$ (which will hereafter be used as $\frac{1}{2}\|\vec{w}\|^2$ for convenience, as both are equivalent), the *primal formulation of linear SVMs* is achieved:

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}\|\vec{w}\|^2 \\ \text{s.t.} \quad & y_i(\vec{w}^T \vec{x_i} + b) \geq 1, \\ & i = 1, ..., l, \end{aligned} \tag{2.5}$$

which is a quadratic program of n variables (the dimensions). However, there are two problems with this formulation: it is extremely sensible to outliers and it will not work for **nonlinearly** separable data. There cannot exist a single data point inside the margin or on the wrong side. For this reason, it is called a **hard margin**. This can, and will certainly cause problems when choosing the best hyperplane. For instance, for the outlier case, consider the example on figure 2.9. The

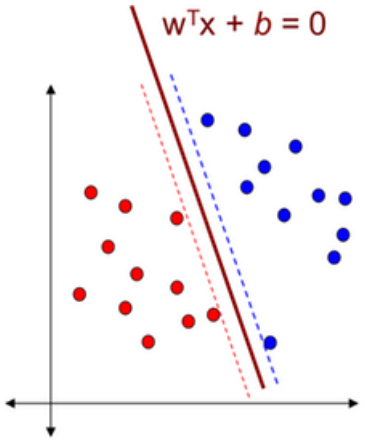


FIGURE 2.9: An outlier's influence on the definition of a hard margin.

outlier from the blue class becomes a support vector, and forces the algorithm to choose a non-optimal hyperplane. To fight this, **slack variables** are introduced into the optimisation problem, turning the hard margin into a **soft margin**. There is a slack variable ξ for each data point, which measures the deviation of an observation from the ideal. In other words, if $\xi > 1$, the observation is on the wrong site of the hyperplane (hence, misclassified); if $0 < \xi < 1$, the data point is indeed on the right side, but inside the margin; finally, if $\xi < 0$, the example is on the right side and behind the margin. Since every constraint can be satisfied if ξ is large enough, there is a need to control its size. So, to control the sensibility of the *SVM* to the outliers, there is a **regularisation parameter** C. The larger the value of C is, the harder it is to ignore the constraints - this causes the margin to be narrow, and if C is large enough the algorithm creates a hard margin. On the other hand, a lower C provides for a larger margin, as the constraints are more easily ignored. The value for C has to be chosen manually, and it is quite a daunting task since there are a myriad of possibilities. A frequent option is to choose it through **cross validation**, which is statistical model validation technique capable of asserting how will a model generalise.

The problem definition is know slightly more complex:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{l}\xi_i \\ \text{s.t.} \quad & y_i(\vec{w}^T\vec{x_i} + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, i = 1, ..., l, \end{aligned} \tag{2.6}$$

Figure 2.10 illustrates the previous example, but now with a soft margin. Notice that while the outlier point is misclassified, the margin is now quite large, thus providing better generalisation. Regarding nonlinear classification, slack variables also play a part. However, this type of classification is only possible due to another "trick" related to the *dual formulation of the SVM*. It is not necessary to approach this formulation formally as it is not

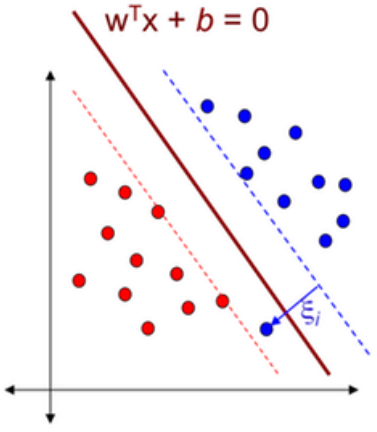


FIGURE 2.10: Classification with soft margin.

needed to grasp the basic theory of the algorithm - it also uses *Lagrange multipliers*, which are not trivial to understand. Be that as it may, the dual formulation uncovers a property of the *SVM* algorithm that, basically, is responsible for its predictive power with nonlinear data: the algorithm needs only the computed dot products between data points, instead of the whole training data. The two most important aspects of this trait are that this way the number of free parameters is bounded by the number of support vectors and not by the number of dimensions (which is indeed good news given the existence of complex datasets with an extremely high number of dimensions which would immediately cause a performance bottleneck), and that the dot product can be replaced by a *kernel*. In the *SVM* context, a kernel function is of the form $K : \mathbf{R}^N \times \mathbf{R}^N \mapsto \mathbf{R}$, and its power advents from its capability of computing dot products in a high-dimensional space $\mathbf{R}^M$ while remaining in $\mathbf{R}^N$. In a more mathematical way,

$$K(\vec{x_i}, \vec{x_j}) = \langle \phi(\vec{x_i}), \phi(\vec{x_j}) \rangle_M, \quad \vec{x_i}, \vec{x_j} \in \mathbf{R}^N \tag{2.7}$$

where $\langle \cdot, \cdot \rangle_M$ is an inner product of $\mathbf{R}^M, M > N$, and ϕ is defined as $\phi : \mathbf{R}^N \mapsto \mathbf{R}^M$. The impact of the *kernel trick* cannot be stressed enough. It is one of the most important properties of *SVM*s, as it provides a solution to Cover's theorem: "A complex pattern-classification problem, cast in a high-dimensional space nonlinearly, is more likely to be linearly separable than in a low-dimensional space, provided that the space is not densely populated" (Cover, 1965). Figure 2.11 depicts the usage of a kernel to uncover a linear decision boundary on a dimensional space higher than original. Note that this is a simple case for demonstration purposes only. Real world problems can have millions of dimensions, which are not humanly visualisable. The most frequently used and known kernel is the **Radial Basis Function** kernel, or **RBF**, but others such as the **polynomial**, **linear** or **sigmoid** kernels are also a common choice. Kernels often have parameters that need manual setting. The *RBF*, *polynomial* and *sigmoid* kernels all require a parameter γ to be defined previously to their usage. The value can be decided through cross validation, much like parameter C from the *SVM* algorithm. In fact, one option that is often resorted to in this case is to perform a **grid search** with cross validation, that is, test the algorithm with different pairs of (C, γ), and then picking the pair with the best cross validation accuracy (Hsu et al., 2003).

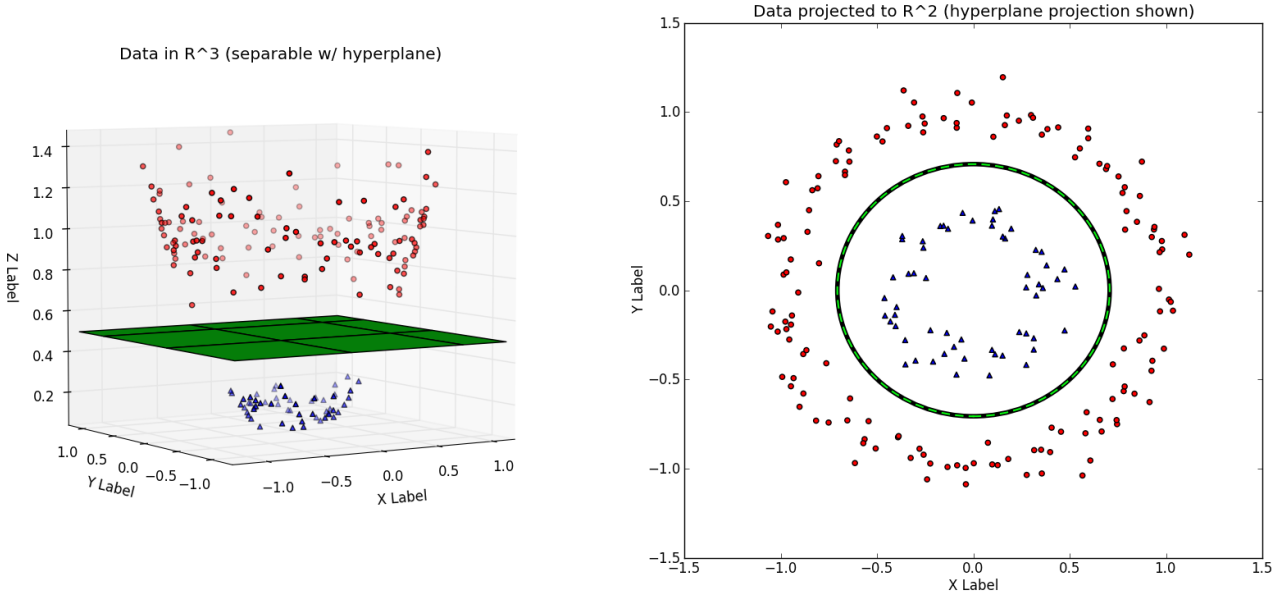


FIGURE 2.11: Although the decision boundary is linear in $\mathbf{R}^3$, it is nonlinear when transformed back to the input space $\mathbf{R}^2$.

With the use of kernels, the formulation of the optimisation problem changes yet again:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{l}\xi_i \\ \text{s.t.} \quad & y_i(\vec{w}^T\phi(\vec{x}_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, i = 1, ..., l \end{aligned} \tag{2.8}$$

When solved with Lagrange multipliers (recall the dual formulation) the decision function also changes to

$$f(x) = sign(\sum_{i=1}^{l}\alpha_i y_i K(\vec{x}, \vec{x_i}) + b), \tag{2.9}$$

where α_i is the *ith* Lagrange multiplier. Figure 2.12 illustrates the same toy example of figure 2.8, now complete with more information regarding the discussed concepts.

The beauty of the *SVM* algorithm is in its relatively simple mathematical basis. In fact, although the models developed can get highly complex, one does not need to be an expert in learning theory in order to understand the algorithm that builds the model, as it can bee seen as a linear algorithm in a possibly high-dimensional feature space nonlinearly related to the input space. Note that no computations are performed in this high-dimensional feature space, as the use of kernels allows for all the necessary computations to be performed on the input

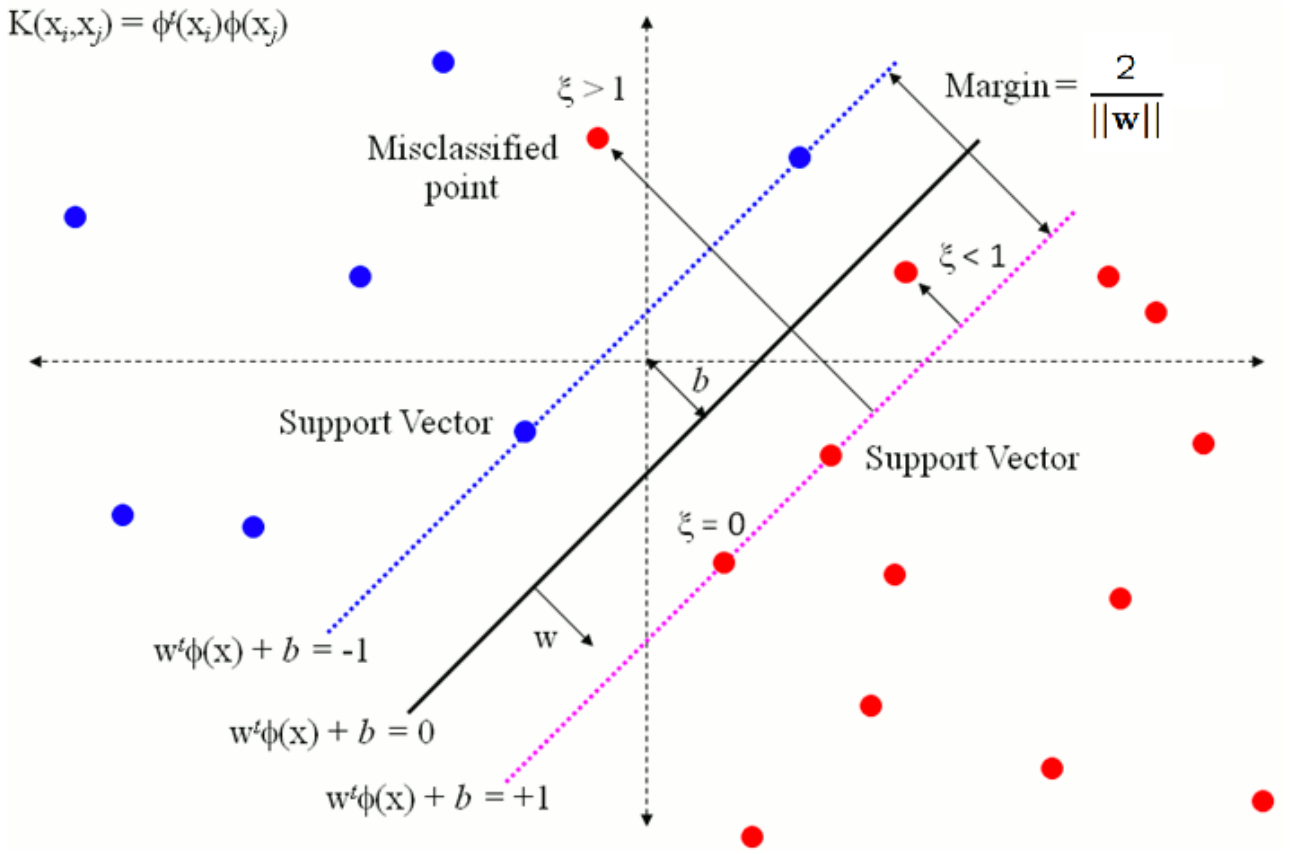


FIGURE 2.12: The same example of figure 2.8, now more complete information-wise.

space. Consequently, in spite of the algorithm actually having a respectable degree of complexity attached to it, one can "pretend" to simply be working with a linear, basic algorithm, and still be able to solve nonlinear problems (Hearst et al., 1998).

Unfortunately, in the anomaly detection area it is not always possible to implement a binary classification process with *SVM*s. The truth is that anomalies are often scarce, while there is an abundance of normal examples. It is not unusual to have two or three anomalous examples per hundreds or thousands of normal ones. In cases like these where one of the classes is **skewed**, a binary *SVM* will likely have trouble correctly classifying observations, as the available information for one of the sample types is too limited. This is actually a disadvantage of the method: the algorithm will not perform well if there is not enough data for training. So the solution is either to acquire more anomalous information or conduct the classification with some other method. However, obtaining more information is not always possible. So, to tackle this problem, methods derived from the standard binary classification capable of working with only **one class** have been proposed. More specifically, the **one-class ν-SVM** proposed in

(Schölkopf et al., 1999) and **Support Vector Data Description**[5], or **SVDD**, proposed in (Tax and Duin, 1999). Although their purpose is similar, the methods are distinctly different. The *ν-SVM* method attempts to find a hyperplane that separates the data points from the origin with the largest margin possible. It defines a small region where it tries to encompass the most observations possible. The quadratic program is

$$\begin{aligned} \min_{w,\rho,\xi} \quad & \frac{1}{2}\|\vec{w}\|^2 + \frac{1}{\nu l}\sum_{i=1}^{l}\xi_i - \rho \\ \text{s.t.} \quad & \vec{w}^T\phi(\vec{x}_i) \geq \rho - \xi_i, \\ & \xi_i \geq 0, i = 1, ..., l \end{aligned} \tag{2.10}$$

and the decision function (again, with Lagrange multipliers so that kernels are usable)

$$f(x) = sign(\sum_{i=1}^{l}\alpha_i K(\vec{x}, \vec{x_i}) - \rho). \tag{2.11}$$

The hyperplane is defined through w and ρ, and notice how there is a ν instead of a C. This regularisation parameter is similar to C, but it is bounded between 0 and 1 (C just has to be a positive value), and has a very precise definition: it sets an upper bound on the fraction of outliers and a lower bound on the number of data samples that become support vectors. So, for instance, if $\nu = 0.5$, at most 50% of the observations will be misclassified, while at least 50% of them will become support vectors. Graphically, this type of classification will be similar to figure 2.10, with the red balls being the anomalies.

The other method, *SVDD*, takes a different approach. Instead of a hyperplane, it uses a *hypersphere*. This hypersphere is defined by a centre a and a radius $R > 0$. It considers normal everything that stands within the radius of the sphere, and anomalous everything out of it. The optimisation problem is

[5]Initially the name was *support vector domain description*, as the title in (Tax and Duin, 1999), but later the authors dropped the "domain" and replaced it with "data".

$$\begin{aligned} \min_{R,a,\xi} \quad & R^2 + C\sum_{i=1}^{l}\xi_i \\ \text{s.t.} \quad & \|\vec{x_i} - a\|^2 \leq R^2 + \xi_i, \\ & \xi_i \geq 0, i = 1, ..., l \end{aligned} \tag{2.12}$$

C influences the size of the sphere - a larger C induces a smaller R. However, this C is also bounded much like the ν in the previous method, but the lower bound is $1/l$ instead of 0. The upper bound is still 1, but it is not a strict upper bound - the algorithm will still function with $C > 1$, although it is pointless since for $C \geq 1$, the *SVDD* searches for a hard margin. A possible example of this type of classification is depicted in figure 2.13: the blue dots are the normal class observations (with one misclassification), the green ones the support vectors, and the red ones the anomalies.

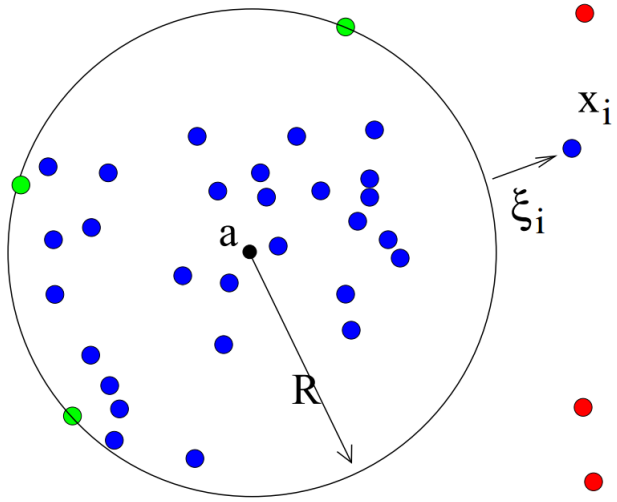


FIGURE 2.13: *SVDD* example.

*SVM*s are nowadays widely used, but even with all the hype around them, they are not the only relevant classification algorithm. Other frequently used techniques are as follows:

Neural networks - A neural network is a machine learning algorithm that can be applied both in a one-class and multi-class settings. A basic multi-class neural network *ADS* operates in two steps (Chandola et al., 2009). First comes the training phase, so that the neural network can learn about all the different normal classes. Second, the testing phase, where the test dataset is fed to the neural network - it considers examples of normal behaviour those test inputs that it accepts, and examples of anomalies those it reject (De Stefano et al., 2000). *Replicator Neural Networks* have been used for one-class anomaly detection (Hawkins et al., 2002), where a *multi-layer feed forward neural network* is constructed with the same number of input and output neurons, which correspond to the features in the data. The training here involves data compression into three hidden

layers. At the testing phase each data instance is reconstructed using the learnt network to obtain the reconstructed output.

Bayesian networks - This machine learning method is used for the multi-class design of anomaly detection. A basic technique for a univariate dataset using *naïve Bayesian networks* estimates the *posterior probability* of observing a class label on a certain test observation. The class label with the largest posterior is the chosen one. The training set is assumed to have independent attributes, and provides the estimations for the likelihood of observing the test instance given class and the class probabilities - the zero probabilities are smoothed with *Laplace smoothing*. It is possible to generalize this method for multivariate datasets by aggregating the per-attribute posterior probabilities for each test observation, and using that aggregated value to classify the observation (Chandola et al., 2009).

Rule based - A rule based *ADS* learns rules to capture the normal behaviour of the system (Chandola et al., 2009). If no rules are applicable to a test observation, it is considered anomalous. These methods exist for both one-class and multi-class designs. For multi-class, the basic algorithm has two steps. The first one is, of course, the learning phase, where it learns rules from the training data using a rule learning algorithm, such as *decision trees*. Each rule has a confidence value associated to it, proportional to the ratio between the test instances where the rule is verified and the total of test instances that the rule covers. The second step consists in finding, for each test instance, the rule that best captures it. The inverse of the confidence associated with the best rule is the anomaly score of the test instance. Another type of rule based technique for one-class anomaly detection, called *associative rule mining*, is defined in (Agrawal et al., 1993), and it generates rules from data following an unsupervised strategy.

Below are some advantages of classification techniques (Chandola et al., 2009):

- Classification methods, especially for multi-class settings, are very powerful;
- The testing phase is relatively fast, since the test observations need only to be compared against the learnt model.

As well as disadvantages (Chandola et al., 2009):

- If the labels for the normal classes are not accurate, multi-class techniques will struggle to provide acceptable results;

- These techniques label the test instances, which is not always the desirable option, namely when meaningful anomaly scores are more important than just knowing the class where the instance belongs to.

2.4.2.7 Clustering

Clustering is a technique whose goal is to define patterns in data by organising the observations into groups or *clusters*. The organisation criteria of the groups is the **similarity** between observations: similar examples will be closer to each other, while not so similar examples will be farther away. Figure 2.14 exemplifies a case with three distinct clusters. This kind of method assumes that the normal observations belong to large and dense clusters, while anomalies will be isolated or grouped in very small clusters (Chandola et al., 2009). It can be used in a semi-supervised setting, by clustering normal data to create a model of the accepted behaviour - if any new instance does not fit into any of the clusters generated or not even close to one, it is considered an anomaly. On the other hand, this method is also a wise choice in unsupervised environments. The reason for this is somewhat intuitive: if the data is unlabelled, there is no way to isolate observations in order to distinguish them. Therefore, through clustering analysis it may be possible to separate the data into clusters, thus providing a safe method of identifying differences between the various examples, or even labelling them. One of the best known clustering methods is the **K-Means clustering** algorithm. This algorithm aims to divide the

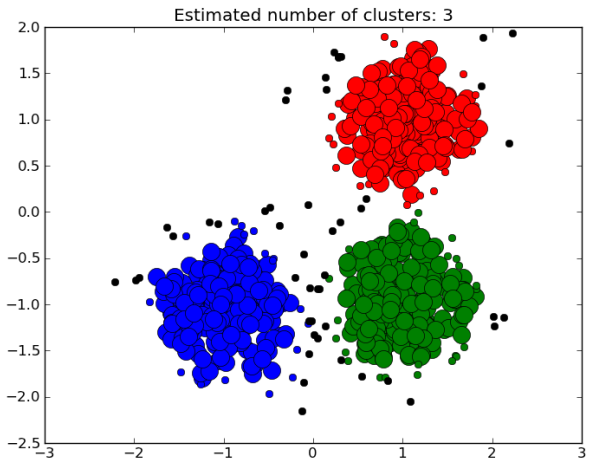


FIGURE 2.14: Clustering example. Black points represent outliers.

dataset into k clusters in which each data point belongs to the cluster with the nearest *mean*. To create the clusters, k *centroids* must be defined in strategic locations, as their positioning dictates the final result - a sane choice is to place them as far away from each other as possible. With the centroids defined, the next stage is take each observation and link it to the closest centroid. When no points remain, new centroids are calculated as the barycentres of the clusters that resulted from the previous stage. At this point, the process starts repeating itself, until the newly calculated centroids are the same as the previously calculated ones.
Besides the advantage of not needing supervised data, clustering analysis is also easily adapted to online anomaly detection. In spite of this, there are some drawbacks: these algorithms tend to be computationally expensive, and if normal observations do not create clusters, the method will probably fail. Also, in high dimensional spaces, the distances between points might be similar, and by consequence the formed clusters might not mean anything useful (Chandola et al., 2009).

2.4.3 Hybrid Approach

Misuse detection and anomaly detection can be combined to form a hybrid system (Depren et al., 2005, Lunt et al., 1992). Both misuse and anomaly systems have their advantages and disadvantages. By bringing both systems together, their weaknesses can be strengthened while keeping the advantages of both sides. The advantages can even be combined to generate new ones. For example, as Patel and Buddhadev described in (Patel and Buddhadev, 2013), their hybrid system will create new signatures for misuse detection through the data analysed by the anomaly component.

2.5 Real World Applications and Prototypes

Intrusion detection has come a long way from a simple log analyser on a single machine to a necessary monitoring and protection system distributed throughout machines on a network. Some *IDS*s made it to the market, while others did not go pass prototyping. Some market *IDS*s and influential prototypes are presented below.

2.5.1 TypeWATCH

TypeWATCH is Watchful Software's second product. TypeWATCH is an anomaly detection system that leverages the power of keystroke dynamics through its biometric properties in order to build user profiles and detect intrusions. Monrose and Rubin presented a set of results strongly supporting the hypothesis of using keystroke dynamics as a biometry in (Monrose and Rubin, 2000) - the typing rhythm, speed, certain habits and behaviours that users demonstrate when interfacing with a keyboard. TypeWATCH arrives years later as a MSc thesis (Ferreira et al., 2011), and it only became more powerful ever since.
TypeWATCH is an *HIDS* with two distinct phases. The first one, called the *enrolment phase*, prompts the user to generate his/her profile. When the user enrols, TypeWATCH starts collecting a set of keystroke metrics such as flight times between keys and dwell times. Note that TypeWATCH **does not** register what the user inputs. A set of these metrics comprise a *sample*, and after thirty samples are recorded, the enrolment phase ends, and TypeWATCH is now ready to enter the *monitoring phase* and detect intrusions: as the user works, TypeWATCH records his/her keystrokes. When a certain keystroke threshold is reached, TypeWATCH compares the newly generated sample against the user's learnt profile stored in the database - If the anomaly score (the difference between the profile and the new sample) surpasses a predetermined limit, the user is considered an intruder, which may lead TypeWATCH to engage in post-intrusion activities - system lock down, sending an email to the user, sending a picture of the intruder or a text message to the user's smartphone, among other possibilities - or simply prompting the user to type in his/her password, as the *IDS* makes a distinction between the severity of intrusions in order to mitigate false alarms.

2.5.2 IDES

IDES (Lunt et al., 1992) stands for *Intrusion Detection Expert System*, and was the first fully functional *IDS* built. Its development began in 1984 at *SRI International* for a government project, and it led to Dr. Dorothy Denning, using her research and development work at *SRI*, to publish her immensely influential work, "An Intrusion Detection Model"(Denning, 1987), in the same year (Innella, 2001).

IDES was an *HIDS* built to monitor logins, command and program execution, file and device accesses, among others, looking for deviations in usage, while also using an expert system to match intrusions to already encoded intrusion scenarios, thus providing a hybrid detection method. *IDES* was later restructured, becoming the *Next-generation Intrusion Detection Expert System*, or *NIDES* (Anderson et al., 1995).

2.5.3 Haystack

Haystack (Smaha, 1988) was a *HIDS* developed in 1988, by the Haystack project at *Lawrence Livermore Labs*, for the United States Air Force. As a misuse *IDS*, Haystack analysed audit data and searched for known attacks. Regardless, Haystack was also capable of anomaly detection, as it could compare user behaviour with their past behaviours and, in a more generic way, infer about the accepted generic behaviour of a particular group of users. The last generation of the technology was released in 1989, and went by the name of *Stalker* (Innella, 2001). Haystack was also incorporated in prototype called *Distributed Intrusion Detection System*, or *DIDS* (Snapp et al., 1991), which could monitor the hosts connected to a network.

2.5.4 NSM

Network Security Monitor (Heberlein et al., 1990) was the first *NIDS* to come to life. This system was deployed at major government facilities, where network traffic analysis could potentially offer a large amount of information (Innella, 2001). The goal of this anomaly based *IDS* was to develop profiles of usage of network resources, and then identify possible breaches by comparing the current behaviour against the historical profile (Heberlein et al., 1990).
Heberlein's contributions with *NSM* were extended to the *DIDS* project, where, along with the Haystack team, he created the idea of a hybrid intrusion detection system (Innella, 2001).

2.5.5 RealSecure

RealSecure is an *IDS* currently owned by *IBM* (IBM). It was created in 1997 by a leading security company at the time, called *Internet Security Systems*, or *ISS*, that was bought by

IBM in 2006. RealSecure is a robust *NIDS* that relies on both misuse detection and protocol analysis (Systems, a,b).

2.5.6 NetRanger

NetRanger was developed at *WheelGroup*. It was a misuse based *NIDS*. WheelGroup was bought by *Cisco* in 1998. With this turn in events, NetRanger was re-engineered and re-branded, and is now known as the *Cisco Systems Adaptive Security Appliance* (Cisco).

2.5.7 Snort

Snort is a free and open source *NIDS* developed by *Sourcefire*, which was acquired by Cisco near the end of 2013. It combines the three detection methods already approached - misuse, protocol and anomaly based - to perform real-time traffic analysis and packet logging on IP networks, in order to detect a wide variety of network attacks and probes.

2.6 Conclusions

Intrusion detection is an extremely vast subject. Since there is still no perfect solution (and probably never will be), the problem is still target of deep investigation and study, and as seen with the survey above, the solutions branch into a variety of methods, that in turn tap into different fields in engineering and mathematics.

From the main intrusion detection types discussed, i.e., misuse and anomaly detection, the latter is the one that fits perfectly into the problem at hands. On the other hand, anomaly detection is also the hardest of the two problems, not only for the inherent complexity of the system's design and implementation, but also for the need to deal with an immense number of false positives (normal behaviour observations that are misinterpreted by the system as anomalies) after deployment. Of course, tuning and tweaking the detection algorithm will certainly help in reducing the false positive ratio, but nothing can fully prepare the system for the real world, as anomalies can appear in ways that are not predictable in a timely fashion by the

engineers responsible for the project.

Based on the explored literature, the preferred type of detection algorithms in problems of this nature seems to be the classification type. More specifically, *neural networks* and *SVM*s. The reason for the preference is simple: given all the constraints imposed by the problem, these algorithms are robust and perform extremely well. Between the two previously mentioned, *SVM*s have the advantage of not consuming too much time in the training phase while keeping the end results as good or even better than the ones from *neural networks*. This, allied to the *SVM* method's algorithmic simplicity, automatically selects it to be the core algorithm of the anomaly detection system to be developed.

Chapter 3

Proposed Model and Architecture

Gathering and organising all the knowledge involving intrusion detection systems was a necessary step that will now allow for a more pragmatic design of the solution for the problem at hands. However, before even starting to think about design details and features, it is essential to be entirely familiar with *RightsWATCH* logging methodology, in order to know what should actually be designed and developed.

3.1 Initial Approach

Any interaction that a user has with information protected by *RightsWATCH* is recorded on a *Microsoft SQL server* **logging database**. This database is composed by a small set of tables, and the full record is stored only into one of them - the other tables are there only for structuring purposes, as it will be seen shortly.

Let the table where the records are stored be called the *main table*. Apart from the primary key - an incremental value assigned to each new record that arrives -, this table has thirty-eight columns, some of which being foreign keys. Part of this set of columns link the table to the previously mentioned (secondary) tables, and the remaining set to other tables in another database. This second database is the **configuration database**, which holds a variety of information needed for configuring different *RightsWATCH* components.

At this point, and after studying the databases, three facts are clear. The first is that, to build

an anomaly detection system based on the *RightsWATCH* logging system, one only needs access to the main table. This table will be the main data source for the system, and by having all the relevant data already gathered, it will simplify the data preparation and selection steps. The second and third facts, however, will hinder the preparation step. The second fact is that, after looking at the data, it became instantly clear that it needs to be cleaned and normalised. As for the third fact, the different data types of the main table will be an obstacle. The records stored have information in binary, numeric and categorical formats, where the last one refers to strings of text. Since most machine learning algorithms only deal with numeric data, a common decision in problems of this kind is to divide the categorical values into their individual values, performing what is referred to as **one-hot encoding**. For instance, a categorical feature whose values are "yes", "no" and "maybe" will disappear, leaving in its place three new binary features that correspond to each of the values. This has its advantages, but the obvious disadvantage is a large increase of features. Both the second and third facts pertain to common, yet important obstacles to overcome, and the implemented solutions will be detailedly described in chapter 4.

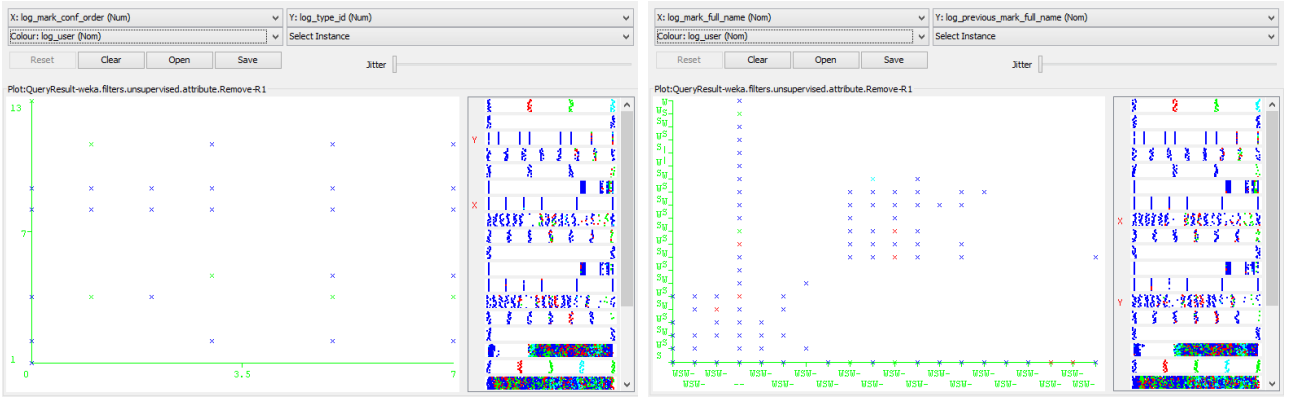
Before starting to develop or even thinking about a complex system, trying to visualise the data in search of evident patterns is always a good idea. Not only this will help to understand the data better, but it is possible to extract certain patterns. In order to plot and visualise the data, a software named **Weka** was used (Hall et al., 2009). *Weka* is an open source data mining software developed by the Machine Learning Group at the University of Waikato, in New Zealand. As the authors describe it, "*Weka is a collection of machine learning algorithms for data mining tasks. (...) Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization.*".

To use *Weka*, one can either input a file with the dataset, or grant it direct access to the database. In this case, the second option was chosen, as it would be faster than building a file with the data in a format accepted by *Weka*. This way, only a configuration file for *Weka* had to be created, so it could know what to do with the data types that it would receive from the database connection. This was necessary as different database management systems exist, with their own features and definitions.

With everything up and running, the data could now be visualised. Immediately, the idea that the data could, in its raw state, provide useful patterns, was discarded. The information

that *RightsWATCH* extracts from user activity is extremely fine grained, which means that no feature by itself can create patterns that can successfully distinguish between users. Even when plotting features against each other, hardly any relevant patterns arise, which also means that the features are poorly correlated, or at least not correlated enough so that the patterns can be seen by the naked eye. In fact, the features that do show correlation between them, are only correlated because they bring the *same information* to the dataset. An example of the raw features' visualisation is documented on figure 3.1.

With this initial (and manual) data analysis done, the data was observed, studied, and it



(a) Most of the correlation attempts look like this. Each colour represents a user.

(b) The most interestingly shaped pattern, which relates the classification mark of a file with the previous mark it had.

FIGURE 3.1: Raw feature visualisation examples.

was shown that one cannot expect much from its raw state. As such, now is the right time to start designing the system. The full architecture of the developed framework is discussed and explained in the next section.

3.2 Architecture

Based on the information on the environment and the problem so far, it was decided that the framework needs to be capable of three different tasks. These tasks are:

1. Reading from the data source and transforming the data into a single dataset that the algorithm can understand;

2. Run the detection algorithm over the dataset and output a classifier;

3. Use the classifier to infer the legitimacy of the new log entries that arise, and output the result in a way that humans can understand and extract conclusions from.

The proposed architectural design of the framework is depicted in figure 3.2.

Since there are three main tasks, it makes sense to divide the workload through three main

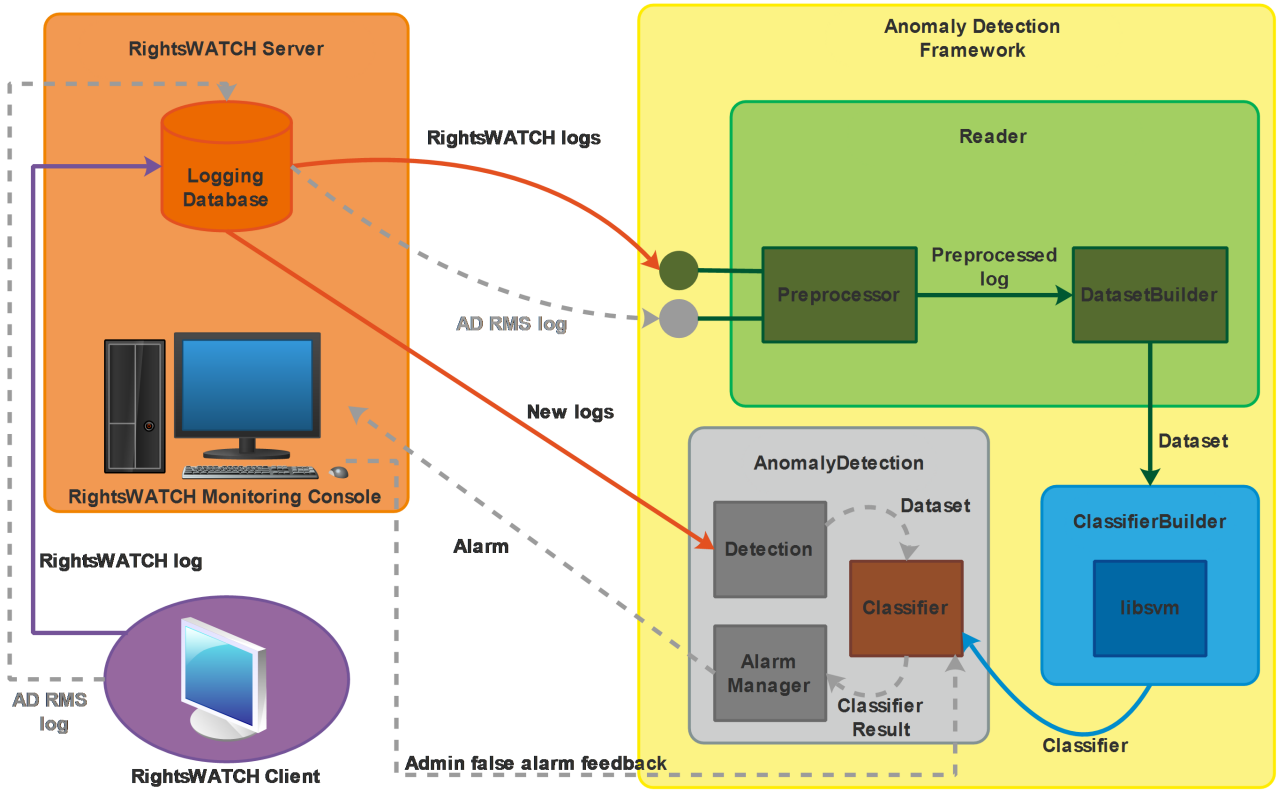


FIGURE 3.2: Architectural design of the framework.

modules, so that the modularisation and encapsulation principles are respected.

The *Reader* module is responsible for reading and shaping the input data. It is divided into two secondary modules. The first module is called *Preprocessor*, and it selectively reads the features that are considered relevant and preprocesses them which, when dealing with categorical data, means tearing the feature apart into n features, where n is the number of existent cases for the raw feature. The raw features that it reads have to be previously selected by the developer, through manual input directly into the code. This way, the framework can easily be adapted to accept other data sources, as one only needs to *teach* this module how to access the data source(s) and what features to use. This is represented in figure 3.2 by the **AD RMS logs**. *AD RMS*, or *Active Directory Rights Management Services*, is an infrastructure provided by

Microsoft, that gathers all the server and client technologies to support information protection through the use of rights management in an organisation, which *RightsWATCH* uses. The arrows are dashed and greyed to indicate that, although possible (and, for this specific case, planned), their usage is not implemented.

After the data preprocessing, this module outputs a file with the result. This result then serves as input for the second module, the *DatasetBuilder*. This is where the features are reorganised in order to produce the final dataset. Besides its basic operation, this sub module also preserves the user's privacy by completely transforming the data into something that cannot be tracked back to its original state - the dataset is fully composed of numeric values, whose relationship to the real data only exists while the file is being created, and later eliminated when the file is ready.

The dataset file will be the input source for the *ClassifierBuilder* module. This module analyses the data and generates a model of the user's behaviour. This model, also known as *classifier*, will be used to determine if future user events are coherent to whatever is considered normal for the user. To perform this step, a detection algorithm is needed. As stated before, the *SVM* method was the chosen one for the task. However, there is no need to implement the algorithm from scratch, since it is already implemented in an open source fashion. From all the implementations available, the **libsvm**, by Chang and Lin (Chang and Lin, 2011), was the chosen one. This library packs the standard *SVM* algorithm along with the most relevant variations, such as *one-class SVM* and *SVM* for regression. Another version of the library, available in the website, also packs the *SVDD* algorithm. The library is already well prepared and optimized to deliver the best results it can. In fact, since its inception in 2001, *libsvm* was successfully integrated and used in various data mining problems (the full list is available on *libsvm*'s website).

To perform the detection, two different *SVM* algorithms were investigated, although one of them to a much greater extent than the other. These algorithms are the *one-class classification* algorithm and the *SVDD*. This second algorithm was only considered in the final stages of the investigation, mostly for performance comparison. Both of these relate to semi-supervised learning, and are the only *SVM* options available for this problem. The reason for choosing semi-supervised algorithms is one of the most important aspects of the problem that ignited this investigation: information about *bad* user behaviour does **not** exist and it is not easy to

idealise due to an infinity of possibilities. This means that the solution is to create a model of good behaviour by using the good data available, and tag everything that deviates from this model as undesirable behaviour.

Every user is a different person, with different habits, behavioural patterns and work positions. As such, it is both unsafe and unreasonable to simply assume without any kind of proof that, to build the behavioural models for each user, the exact same algorithm with the exact same parameters will produce the best possible model. Moreover, this hypothesis cannot be tested in this investigation, since all the data corresponds to only four users, which is not enough to allow for a definite conclusion. Given all this, the secondary function of this module is, in the validation stage and following a conservative approach, to test different variations of the detection method. More specifically, to test the two *SVM* algorithms, each through four different kernels - *linear*, *RBF*, *polynomial* and *sigmoid* kernels - while performing a grid search over the algorithm parameters - C for *SVDD* and ν for *one-class* - and kernel parameter γ (except, of course, for the *linear* kernel, that does not use γ). Various classifiers will be generated, and the one with the best performance will be chosen. This process is described in chapter 4.

The final model will be stored and used by the *AnomalyDetection* module, which is divided into three sub modules, *Detector*, *Classifier* and *AlarmManager*. The first one is responsible for retrieving new logs generated by user activity, extract the features and prepare them for the second sub module, which evaluates the entry, sending the result to the *AlarmManager* sub module. This last one will react to the result, issuing an alarm to the administrator with the corresponding threat level. Shall it be the case of a false alarm, the administrator can mark the event as benign, triggering a mechanism which will send this feedback to the *Classifier* sub module, allowing it to accomodate to this information in order to avoid similar mistakes in the future.

Of all the three main modules, *AnomalyDetection* is the only one that is not fully implemented. Although the *Classifier* sub module had to be ready as it was needed for verification and validation of the solution, the other two are only prototyped. Indeed, this module is the one with the least relevance to the investigation. Also, with all the emphasis on the other two and with the problems that arose over time while developing and testing them, in the end there was no time left to finish this module. These problems are clarified in chapter 4.

Chapter 4

Experimental Setup

4.1 Testing and Development Environment

The *ADS* framework was developed at *Watchful Software*'s office, so that the *RightsWATCH* team could provide the necessary guidance to allow a smooth future integration of the framework into *RightsWATCH* itself. In the same way, all the testing was performed there, with data from four of *Watchful Software* collaborators (the reason for this number of collaborators is explained in section 4.2.3). The quality of the logs available is discussed in section 4.1.1 - it will only be briefly discussed however, since there is no need to go deeper as the type of information itself will already be explored in the next section - and the technologies used to develop the framework are approached in section 4.1.2.

4.1.1 RightsWATCH Users

As it was said before, the framework was tested on a set of logs that stems from four different users. These four users are responsible for customer support, product marketing and sales, and do not perform tests on the software during their normal workflow. Although, in critical situations such as the release of a new *RightsWATCH* version, it is possible for some of them to be asked by the development team to test the application's features, which may inevitably translate into outlier records on the database as they will be performing tasks that would not

perform otherwise. Fortunately, the work that these users normally do already involves using *RightsWATCH*, so in these critical moments they end up either with minor testing tasks or, in the best case, testing the product through their normal workflow, thus generating logs that do not deviate too much from what it is considered normal for them. The available number of logs for user 1, user 2, user 3 and user 4 are, respectively, 5714, 3120, 2514 and 2365.
Having only four users is obviously not enough to perform some specific types of tests such as load testing or bottleneck identification. However, it is impossible to gather data from more sources, since client information is off limits and the option of generating artificial data would consume too much time. Alas, four users will have to suffice for the goals of this investigation.

4.1.2 Technologies

The technologies used to create the framework were chosen within the context of *RightsWATCH*'s development, by integration reasons. As it was already mentioned, the logging information is stored on a *Microsoft SQL Server* database. The new table created for the framework, *LogTraining*, is in the same database, under the same *DBMS* technology. The communication with the database is done in the *Reader* module (see figure 3.2), which is fully implemented in **C#**. The reason why this language was chosen is because *RightsWATCH* also uses it for the same purpose. This way, besides maintaining code coherency (different components, with the same purpose, written in the same language) by the time the framework is integrated with the product, it is possible to make use of the **common components** code that was already developed in *RightsWATCH*, thus saving time on the development phase by not "reinventing the wheel".
The *ClassifierBuilder* module, on the other hand, is implemented in **C++**, serving as a *wrapper* for the *libsvm* library, which is written in **C\C++**. Certain parts of the *libsvm* library were slightly changed, in order to parallelise operations with **OpenMP**. All the code was written in **Visual Studio 2013**.

4.2 Dataset Creation

"Sometimes it is not who has the best algorithm that wins; it is who has the most data." - Well known aphorism in Machine Learning.

The creation of the dataset for the detection algorithm to use as input is not a trivial task. Practical experience along the data mining field suggests that the data preparation stage takes about sixty to eighty percent of the time involved in a data mining problem (Romano, 1997). In fact, organising the data in an efficient and useful way is one of the most important steps in problems of this genre. Not only retrieving the data from its source(s), but also picking the best features or even hand-engineering new ones. Of course, if the data has no quality in terms of the information it holds within, the algorithm will not perform well. On the other hand, quantity is also important - the more observations there are, the better. The common problem with this is that more data might be hard to obtain: it might be expensive; it might even be impossible; and in supervised environments, for instance, it might require a large amount of time analysing every observation in order to label it. This is why all efforts must be made to take advantage of all the available data, carefully studying it so that whatever gets discarded as useless is, in fact, useless.

4.2.1 The Horrors of Industrial Databases

Nowadays, companies store practically all their relevant data on databases. To data mining researchers, these real world industrial databases can be considered one of their worst nightmares, and the reason is simple: real data is, most of the times, dirty and cluttered, and the databases are not prepared for data mining. This is particularly seen in databases comprising non-functional (i.e., informative) information, such as usage logs.

When a software project starts, the initial database schemas are created. Assumptions are made, as well as decisions grounded on those assumptions. Time passes and the project grows, and inversely proportional to this growth is the maintenance and attention given to the database systems. This does not mean by any chance that the people involved in the project stop caring about the databases, as they are generally essential to keep the project up and running, or at

least part of it. The reality is that even with noisy or messy tables, and little or no intelligibility of the data, the project will still work perfectly. In spite of this, the database will, in time, morph into a more obscure version of its former self. In other words, an array of *convenient* fixes start to change the database, so that the engineers involved in the project can avoid working directly on the tables, which is always a delicate task. Of course, this has nothing to do with conveniency when it comes to data mining. For instance, new columns might be added to the already existing tables, increasing their dimensionality and complexity, when dividing individual tables into more than one table would keep the information clean, organised and easy to understand.

As time goes on, the records might start to become inconsistent and/or incoherent, due to changes in the project that probably are related to higher level changes such as the inception of new features, bug injection and/or correction or changes in the project's business model. Records might begin to be saved with incomplete information, or *missing values* as they are normally referred to in the data mining world. This can happen when values do not apply to a certain case, or they can simply be missing. Not only missing values, but data saved with errors due to bugs or undefined behaviours, noise, or conflicts between values are also common. As it was already stated (although in other words), the database's "health" regarding the quality of the information it holds is usually sacrificed so that the project can continue to evolve, as the engineers devote their attention and efforts to the project's maintenance and evolution at a higher, functional, most profitable level. The above mentioned problems usually have minor side effects. These side effects are more commonly noticed on secondary tables, that are not crucial to the project's functionality (which is the case of *RightsWATCH*'s logging table). In the end, it is highly unlikely for an organisation's database system to be ready for data mining processes.

That being said, there is a need to prepare and cleanse the data, in order to remove noise, inconsistencies and incompleteness, which might disguise possibly useful patterns. The next sections clarify all the steps that were taken in order to produce the dataset.

4.2.2 Data Preparation and Cleansing

The *main table*, as it was called in the previous chapter, is called *LogItems*, and has all the relevant information for the initial dataset gathered in it. However, information from the other two tables of the database, *LogTypesDictionary* and *LogStatesDictionary* will also be used. An explanation for this will be given in the next section.
The database schema is detailed in appendix A. Before explaining the kind of information that the records hold, it is necessary to shed some light on the way *RightsWATCH* enforces data protection. To put it simply, *RightsWATCH* allows a user to be linked to more than one *company*. Within those companies, the user can choose the *scope*, which can be understood as different company sectors or projects, and within the scopes it is possible to select the *level* of protection. For now, let the focus now be on the analysis of the main table. With its thirty-eight dimensions (excluding primary key), the table stores records that log detailed information about user behaviour. Each records logs a variety of information, such as:

- if the user was connected or disconnected by the time of the action, as well as the type of action - read, write, document classification, email sending, among others -, and the time of the action (both client and server side);
- The combination of company-scope-level used for the action and the previous combination of the previous interaction with the same information, if applicable;
- The plugin that was used, for example, *RightsWATCH* for Office, *RightsWATCH* for Outlook, etc;
- The path of the file that the user interacted with, and the previous path, if applicable;
- The user's machine name, as well as, in case of an email, the email address used to send the information and the corresponding recipients.

Several problems might affect a database prior to the *KDD* process, and *RightsWATCH*'s logging database is no different. *LogItems* has some of the most common problems. The most prominent one is that every record appears to have missing values, be it a blank space or *null* values. Of the thirty-eight table columns, twenty-five of them have missing values, although

one of the columns is a legacy artefact and not currently being used. After a discussion with *Watchful*'s development team, it was concluded that blank space is used where no value is applicable, for instance, when a user works on a document that was previously unprotected, and thus there is no previous directory path for the protected information. As for the *null* values, they are used when a value is not supposed to exist, for example, when a user works on a protected document and all the email related fields of the record get a *null* value. Then, instead of missing values, both cases can be interpreted as *default* values, when there is nothing else to insert on the fields.

Another serious problem is the inconsistency of information over time. As the product evolved, the information in one of the columns also changed. This means that, at some point, previously unseen information meant the same as old information. This can obviously turn into a problem for any detection algorithm, especially if the information keeps changing. This change in the way information is represented happened both due to bug injection and the implementation of new features in *RightsWATCH*.

Regarding the quality and quantity of information, one main issue was identified, and it has to do with the columns related to email recipients. Instead of being associated with the corresponding record on another table, the emails are all piled up in the *LogItems* table, under the form *email1;email2;...;emailn* for n emails. This is valid for every email recipient column, i.e., the "to", "cc" and "bcc" columns.

To solve these problems, various methods can be employed. For missing values, common options are to substitute these values by the mode if it is a categorical dimension or mean if the values are numeric. More conservative approaches choose to completely remove the records with missing values, which has the obvious drawback of reducing the dataset. In this case however, since the appearance of these values is somewhat controlled, the solution is similar for both blank spaces and *null* values. For the former, it was decided to replace the blank spaces with zeros when dealing with numeric fields, and leave them as blank spaces in the remaining cases. As for the *null* values, the decision was to handle these morphing them into zeros when working with numeric values, and to blank spaces when handling the remaining fields. The value zero was chosen because, when used in other fields, it meant that nothing was supposed to exist in that case. In result, the data means exactly the same as it meant before, but it is now more normalised.

The data inconsistencies, as stated before, were identified in one column only. This column is called *log_extra_data*, and just like the name suggests, is responsible for logging data that does not fit in any other column of the table - its purpose is actually the reason for the data inconsistencies. In a normal situation, the best solution would be to divide all the information throughout new columns. Nevertheless, this is not the case, since that under all the data entanglement in the column's values, only one of them is of interest. This value, called *rule id*, only appears when the user applies a rule, which is not always the case. Moreover, this value disappeared from the logs for a considerable amount of time (months) even when the user did apply a rule, due to changes in the code that handles the logging process. This bug is now corrected, and the solution for the overall inconsistency goes through extracting and using only the rule id, leaving a blank space when it is not present. In spite of this, given the bug that caused the value to disappear, it was decided to leave the rule id out of the initial dataset for now, as this value, by itself, is the only one that has real missing values. Its reintroduction on the dataset will be evaluated in the future, when more correct instances of the value are available. In fact, the more correct observations of this value are available, the less impact will the missing instances have, as the specification of the framework dictates that the detection algorithm has to be able to evolve and learn continuously from new information which, with the bug fixed, turns the rule id into a valid and possibly valuable dimension again.

Finally, the email problem was solved through the creation of new tables. These tables are called *LogEmailsX*, where *X* can be one of three possibilities: *To*, *Cc* or *Bcc*. These tables associate, for their respective type of recipient, the email addresses with the logs from the main table. In other words, each table has a *log_id* as a foreign key (which is, in fact, *LogItems*' primary key), and each of their records corresponds to the email addresses for a given log. For instance, if log number seven had two emails on the *to* field, the *LogEmailsTo* table would have two records with *log_id* equal to seven, one for each address.

All of these changes could not be directly implemented over the *LogItems* table, as it would directly affect *RightsWATCH*'s development. Instead, a new table, called *LogTraining*, was created. This will be the main table for developing the framework, as it will have *RightsWATCH*'s logs in a normalised, more refined version. Some columns of this table will in fact be different from the original table, as it will hold the raw *learning features* instead of the raw initial information. With the data prepared, data selection and feature extraction are the next phases of

the process.

4.2.3 Data Selection and Feature Extraction

The data selection process can turn out to be complex and exhausting. In this case, fortunately, it turned out to be quite straightforward.

The database available for this investigation gathers all *RightsWATCH* usage information for the *Watchful* team. This means that logs generated by both software and quality assurance engineers are also included, which actually represent a significant forty-eight percent of all the available data. The problem here is that they generate most of their logs when testing features and/or correcting bugs. Sometimes, this means logging dummy information, which is completely different from normal. Of course, this is their usual behaviour, and they also use *RightsWATCH* normally when they need, so one might argue if their logs are or are not valid for the task at hands. However, *Watchful* follows the *SCRUM* software development methodology which, in this case, means that a developer after executing a task (behaviour pattern), might never actually repeat it or perform something completely different afterwards. In the end, the behaviour profile of a developer would not be stable enough for a trustworthy anomaly detection.

To account for this issue there are not many possible solutions other than either select the valid records or to completely eliminate all of them. Unfortunately, time is a valid resource, and there is not enough of it to analyse each log individually. Consequently, the decision here was to eliminate all the developer's logs, thus reducing the database from eighteen to four users.

Concerning feature selection and extraction, two methods were used. The first method is an heuristic one, as it simply involved human thought on feature value. The second method was then used to cover up the first one's flaws, as it formally rates the features according to their relative importance (variance), through the use of the *PCA* algorithm.

4.2.3.1 Heuristic Feature Extraction

In situations similar to this one, human opinion about the data matters. Despite all the efforts, feature extraction is still an NP-hard problem, and although algorithms nowadays are capable of uncovering important features where they are not expected, they can fail when dealing with the most obvious cases. So, before handing the data to an algorithm and waiting for its output, it is always a good idea to first look at the data and try to isolate possibly valuable features.
A part of this heuristic feature extraction was already performed when rule id was extracted from the extra data column, even though it will not be used while this investigation is conducted. Another part was done when building the *LogTraining* table. The final version of this table has twenty-four columns (excluding the *log_id* column which was kept in order to associate this table to the email tables that were also created), where one of them corresponds to the label of each log (anomaly or not). Since its first version, the changes were rather subtle - the removal of rule id and the addition of the path features and three existent id fields in *LogItems*, as it will be explained below. The final version is presented in figure 4.1 (where it is associated with the email tables). As it can be seen, some columns are exactly the same as they were in *LogItems*, with the difference that there are no missing values in this table. The email recipient columns have disappeared, as they were stored in the new email tables. Seven columns were left aside: *log_server_date*, *log_read_only*, *log_extra_data*, *log_guid*, *log_previous_guid*, *log_email_subject* and *log_email_header*. Both the "read only" and "email header" columns are not being used by *RightsWATCH* at the present. The "extra data" column was discarded since, as already stated, only the rule id was needed and it was excluded for now due to possible errors on the classification, given the bug that affected the value. The *guid* columns refer to a property of protected data of that same name - which stands for *Globally Unique Identifier* - meant for information tracking. Each time the data is target of user activity, it receives a new *guid*. These fields are dependent of the data the user is working with and, accordingly, the user logs will show a different value in these columns every time he/she works with a different document or email - they are not directly related to the user's behaviour and thus were excluded. Also excluded, for the same reason, was the "server date" column. Finally, the "email subject" column was the hardest one to decide whether to retain or to exclude from the dataset. Indeed, email subject might be important in data leak detection, but in order to use it the system would need to have some sort of dictionary, that would

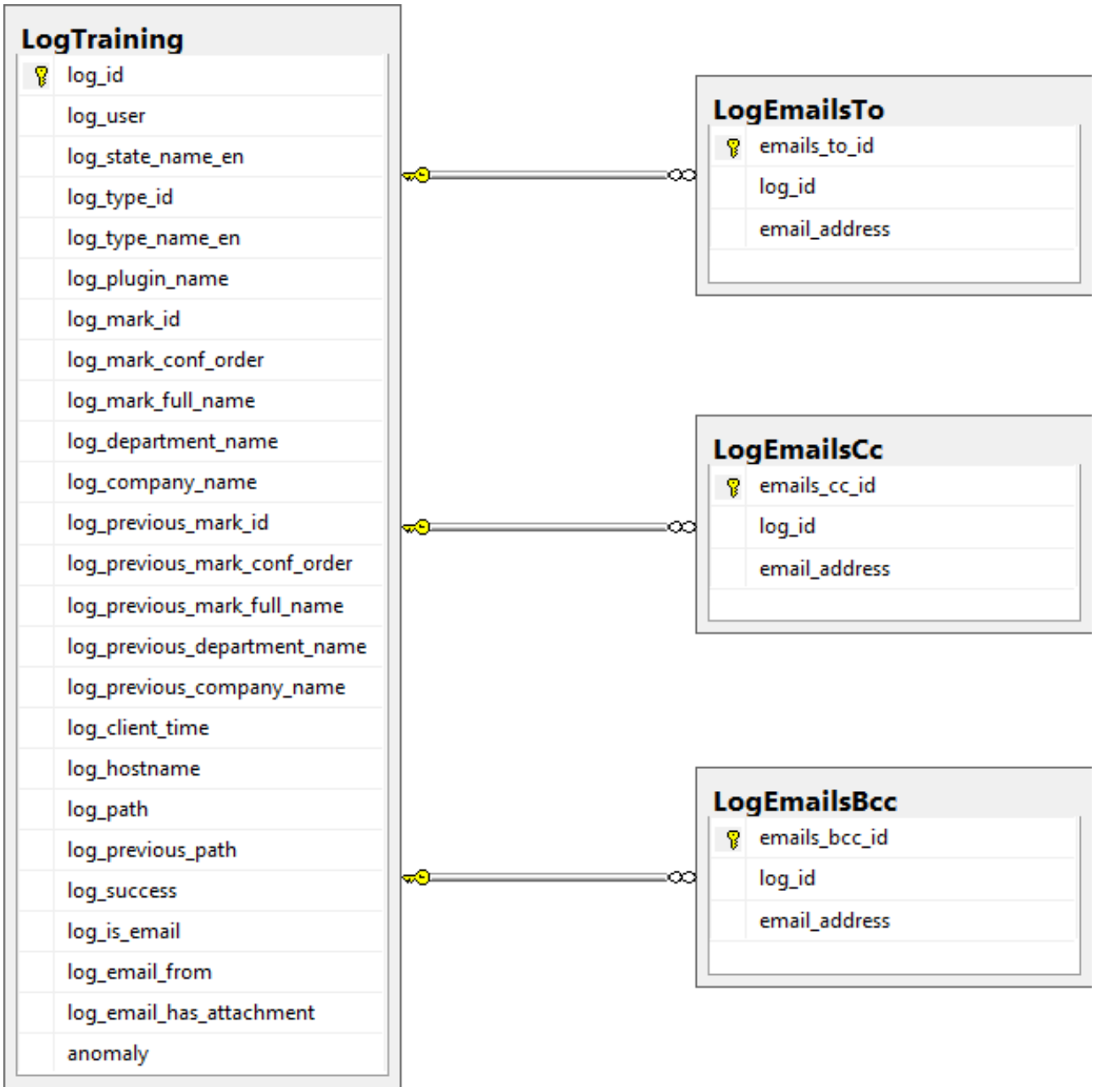


FIGURE 4.1: *LogTraining* schema.

gather all the words from the email subjects in the existent logs and use them to search for anomalous word combinations or even unknown words. In spite of the fact that a system of this kind can be complex and/or time consuming to implement, it would be built only for this column in particular. So, it was decided to leave this column out of the dataset, at least in this investigation's time frame.

Two categorical columns were discarded for not bringing any new information to the dataset. These were *log_mark_name* and *log_previous_mark_name* - there is no need to have these when

there are already two other columns with the mark's full name. Other two columns, *log_path* and *log_previous_path* were initially removed due to the conviction they introduce too much noise. However, they were later reinserted since the reason of their removal had no solid ground to stand on. Most of the *id* columns were also discarded, being that the quantity of information they bring is already assured by the other related (categorical) columns, with the advantage of not risking the detection algorithm to induce some kind of numerical order on the values. The two *mark_id* columns were initially discarded as well, but were later brought back as they were needed to find out the number of document encryptions. The reason for this will be clarified in section 4.4.

The column *log_email_item_id* from the *LogItems* table had its data transformed and its name changed. As it was, this column had a numeric value that is possible to associate with the *guid*, when a user attaches something to a protected email. When there are no attachments, the value is *null*. The information of this column is somewhat related to the user's behaviour, in the sense that it is not *null* when the user attaches something to the email. Therefore, a new column called *log_email_has_attachment* was created, and it holds the information that matters from the original column with only two values: "0" for all the *null* occurrences (recall the missing values solution) and "1" for every item id. The actual *ids* were discarded since there is no need to know them, but rather to know if the email had any attachment or not.

Two other *id* columns from the *LogItems* table, *log_type_id* and *log_state_id*, are foreign keys to the other two original tables of the database, *LogTypesDictionary* and *LogStatesDictionary*, respectively. In *LogTraining*'s case, however, they are primarily used in the table construction query in order to fetch the *log_state_name_en* and *log_type_name_en* columns from their respective tables via SQL *join* operations. The *log_type_id* column was initially used for some time due to an idea born from a brainstorming session with the *Watchful* team, where it was noted that it is of *Watchful*'s interest that the system can somehow rate each log through the relative danger of information disclosure it poses. This was named **danger score**, but it was later discarded and will not be addressed in this investigation given its irrelevance regarding the final results.

Any feature extraction process would not be complete without somehow testing the data as it is extracted and studied. Although at this point the *ADS* framework was not yet ready to

accept data, treat it, and feed it to a classifier, *WEKA* is already distributed with the necessary tools in order to do so. In fact, *WEKA* also comes with a wrapper for *libsvm*, which means that one needs only to specify the path for the *.jar* file (*libsvm* comes in a variety of formats) at *WEKA*'s configuration file, and leave all the hard work to the program. Be that as it may, the tides can turn when it comes to setting up *WEKA*'s connection to the database, as configuration steps must be followed both in *WEKA*, by setting up the database connection configuration file according to the database type, as well as choosing the conversion standard for each data type, and in the operating system, by creating the type of connection through some administrative tool (*data sources*, in Windows). The fact that the configuration varies with the database type and operating system, allied to the not so clear *WEKA* documentation, this process can easily backfire. Note that *WEKA* also accepts its input through files, but this would imply that the data had to be converted into the proper format.

The first test was done with the very first version of the dataset - let it be called the *rule id version*, as it is the only one that has this feature. The second test used the same data but without the rule id variable. The third test, while not having the rule id as well, introduced the path features.

As there are no anomaly examples, the ideal would be to set *WEKA* to use one-class classification. Unfortunately, this is not possible to achieve with this dataset in particular, unless using only the data from a single user. Since there is no way to formally prove the capability of the classifier to detect anomalies, successfully distinguishing the user from other users is the closest thing one can get at this point. At a first glance, this can be easily achieved by generating a classifier model with the one-class classification algorithm, and then using that model to classify the observations gathered from other users. The problem is that, as the *SVM* algorithms needs categorical data to be converted into binary data, *WEKA* starts by one-hot encode the features into their respective feature subspace. So, given the nature of the dataset, some users will then have features that other users do not (the most obvious example are the path features). As such, *WEKA* will immediately complain of incompatible datasets. The workaround for this was to use the standard *SVM* algorithm, that takes more than one class: Although this algorithm does not fit into the problem, it would, in any case, help understanding the data.

For this test process, two *SVM* types were used, and the default values for the *SVM* execution

were left untouched. The first *SVM* type was **C-Support Vector Classification**, or **C-SVC** (it is the common *SVM* algorithm in *libsvm* for two-class classification), with the *RBF* kernel and $C = 1$. The second type was the *one-class SVM*, also with the *RBF* kernel and $\nu = 0.5$. The testing logic was as follows: firstly, the observations of a user were picked as standard observations, while the rest of the observations (from the other users) were marked as anomalous. Afterwards, a dataset with observations from the "legitimate" user and one of the "anomalous" users was handed to *WEKA* in order for it to create the classifier using the C-SVC method. The classifier was created using 10-fold cross validation. For *WEKA*, this means performing the usual cross validation algorithm (in this case, ten times) on the training set as a way to get an average result of all the classifiers, and then running the algorithm an eleventh time, but now on the whole dataset. This last run is what generates a classifier model that can be deployed in practice. Finally, the process is repeated for the remaining two anomalous users, and then another user is picked as the one with the standard logs, which starts this process all over again, but excluding the user combinations that was already made, as the final result will be the same even if their positions are inverted.

As a final test, each user is tested against himself with a classifier built using one-class classification. As a side note, the username and hostname features had to be removed from the datasets - as they are unique to each user, the classification accuracy of the two-class *SVM* would always be a hundred percent. The *log_email_from* feature was also excluded, since it can also serve as an unique identifier for the users who use only one email address, and the client time feature was fetched from the database in such a way that only the hour is retrieved.

Tables 4.1, 4.2 and 4.3 contain the C-SVC results for the first, second and third test respectively. Table 4.4 aggregates the one-class classification results for each user, regarding each test.

Each table exposes the correctly classified instances percentage between the four users' datasets. Of course, these values can be completely misleading. For instance, a classifier that just predicts legitimate actions, whatever the example type may be, will still have ninety percent of classification accuracy with a dataset composed of nine hundred legitimate cases, and a hundred of anomalous ones. Certain measures exist to counter this misleading effect and evaluate the classifier's quality with more precision. These will be used when actually testing the *ADS* framework, on section 4.3.2.

Looking at table 4.1, and keeping in mind that the dataset is not that different from the original, the results are not so bad. The corresponding values for the one-class classification (table 4.4) are lower due to the ν value. With a lower value, the percentages would be higher, but the classifier would also be less strict towards classifying samples as anomalous, which means more false positives. Regardless, this is not a concern for now.

On table 4.2, green coloured values correspond to values that are higher in comparison to

	Classification Accuracy Results			
Anomalous / *Legitimate*	User 1	User 2	User 3	User 4
User 1		67.7%	79.5%	71.1%
User 2			70.9%	67.5%
User 3				73.1%

TABLE 4.1: Classification accuracy results for the first version of the datasets without the path features.

	Classification Accuracy Results			
Anomalous / *Legitimate*	User 1	User 2	User 3	User 4
User 1		67.7%	78.9%	70.8%
User 2			70.6%	67.6%
User 3				73.2%

TABLE 4.2: Classification accuracy results for the version of the datasets without the *rule id* feature and still without the path features.

	Classification Accuracy Results			
Anomalous / *Legitimate*	User 1	User 2	User 3	User 4
User 1		64.7%	77.6%	70.7%
User 2			68%	63.5%
User 3				71%

TABLE 4.3: Classification accuracy results for the first version of the datasets with the path features.

the previous table, and red coloured values correspond to the ones that are lower.

The majority of the results suffered a slight drop. Fortunately, the differences between both cases are almost negligible, which did not influence the decision to remove the feature for the reasons already explained.

	Classification Accuracy Results		
Feature sets / *Users*	1	2	3
User 1	49.1%	49.5%	49.7%
User 2	50.4%	50.1%	50%
User 3	51.5%	46.6%	52.5%
User 4	49.9%	49.8%	49.9%

TABLE 4.4: Classification accuracy results for the users themselves.

Following the same logic on table 4.3, it is shown that all values are lower than their counterparts in the previous table, and this might be due to noise generated by the addition of the path features. Indeed, at this point one can only guess, as it is impossible to know for certain if this is the case. However, as seen in table 4.4, the classifier performed slightly better when evaluating logs from the user it refers to. As such, the path features were still allowed to stay in the dataset until further evidence of their degrading effect on the quality of the data.
The classification results can aid in assessing the overall quality of the data, but they cannot provide any deeper insight on the features themselves (unless, of course, if every feature is temporarily removed, and classification is done with the remaining ones, which would take quite some time to achieve). The heuristic method of feature selection, along with the preliminary tests that were shown, did help to transform the most obviously noisy/useless cases into something more useful, but with these alone it is not possible to ensure that the tables only contain valuable information, as it could be exemplified with the path features. To aid in this validation attempt, the *PCA* method was employed.

4.2.3.2 A More Formal Approach - PCA

The Principal Component Analysis method is commonly used for two reasons. One of them is *visualisation*: as the *PCA* process converts the observations into the principal components' feature space, the once dull and apparently meaningless plotting of features is replaced by possibly interesting patterns (clusters) that are achieved by plotting the principal components against each other. It also allows for the visualisation of the correlation values between features, through the correlation matrix.

The other reason is the one for which the *PCA* method is most known for - *data reduction*. Apart from reducing the complexity inherent to dealing with large quantities of data (memory usage, disk space needed, etc.), reduction also helps speeding up the learning algorithm while giving the researcher some insight on the data's nature, through the ranking of its composing features from the one(s) with the most variance to the one(s) with the least. Unfortunately, *WEKA* does not order the raw features by their importance in an explicit fashion. Instead, one has to do it manually, by looking at the principal components with the higher variance and, from there, examine the coefficients attached to each feature. The higher the value, either positive or negative, the higher the feature correlation with the *PC* - although one still has to decide what does "high" really stands for, which depends on the purpose. Also, this correlation between features and *PCs* can be seen when plotting *PCs* against each other. This way, although not with absolute certainty, the researcher can get a *feel* of which features are the most important, possibly even removing those that are neglectable.

In spite of this, no more features will be removed apart from those that already were. The reason for this is also the main reason for conducting a *PCA* analysis on this investigation: as there is no objective function - i.e., there are no anomalies - it is not possible to know for sure which features should be used. Thus, *PCA* will allow for an deeper insight about the most important (and/or influential) ones, through their variance.

A principal component generated by *WEKA* has the following naming format:

$$\begin{aligned} PC_j &= \sum_{i=1}^{m} c_i x_i \\ j &= 1, \ldots, k, \\ k &\leq n, \end{aligned} \tag{4.1}$$

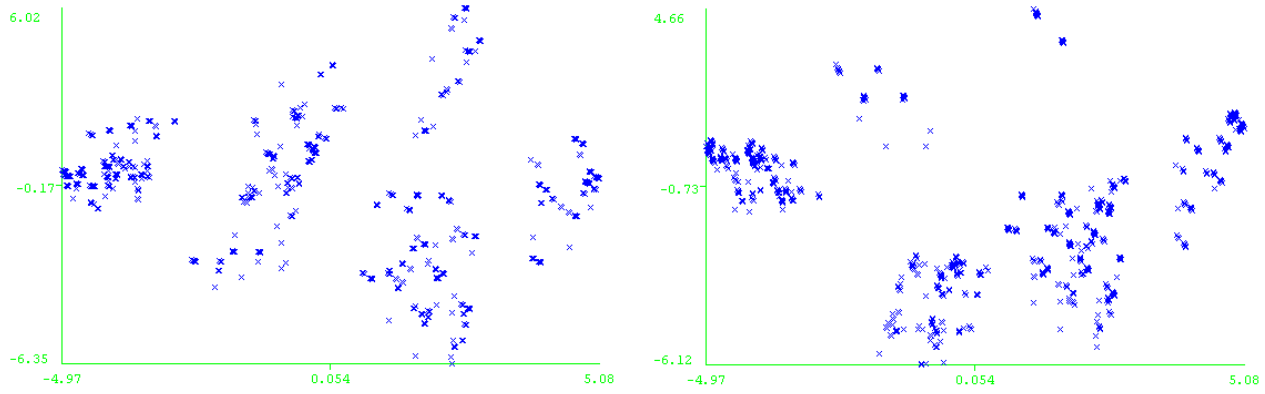
where x is the feature's name, c is a real number representing the correlation coefficient of the feature, m is the number of features used, n is the number of original feature dimensions used on the particular linear combination and k is the total number of principal components.

In this investigation, *PCA* was used over three different datasets. The first one had the base features already described, excepting the paths and the emails. The second one is similar, but only without the emails, and the third one had both feature sets. The reason for three different setups was a suspicion that arose immediately after the first contact with the logs: using the

file paths and all the email addresses is highly likely to introduce too much noise on the dataset, possibly lowering the classifier's efficiency. Part of this suspicion was already (more or less) confirmed with the preliminary classification tests done earlier. So, apart from evaluating the quality of the data, it was expected that by running *PCA* on these three datasets, the validity of this suspicion would be clarified - and it was.

Let the datasets be known as **A** (the one with no file paths and no email addresses), **B** (the one with paths but no email addresses) and **C** (the one with both the paths and addresses). All three datasets are composed by 5714 records, but the number of features varies between them. Dataset A has only 85 features. In dataset B, this value rises to 1554 by adding the file paths, and in dataset C it goes even further by adding 858 email features, summing up 2412 features in total. Regardless of the dataset, *PCA* was executed with the parameters defined to *standardise* the data - which means that the inherent distribution becomes a **standard normal distribution**, i.e., $N(0, 1)$ - and to cover up to 95% of the total variance. All three datasets belong to user 1, as he is the one with the largest number of logs.

The first *PCA* run was on dataset A. The final number of *PCs* for this dataset was 39. Figure 4.2 demonstrates some of the resulting patterns. These were built plotting the top three



(a) Example pattern of dataset A.

(b) Yet another example of the same dataset.

FIGURE 4.2: *PCA* clusters from dataset A. Plotting the *PCs* against each other, starting from the most important ones and going down from there, will output similar patterns that slowly degenerate to ellipsoid, almost linear clusters.

most important *PCs*: (a) was obtained by plotting *PC* one against *PC* two, and (b) by plotting *PC* one against *PC* three. The selected *PCs* represent about 28.85% of the total variance. Due to the sheer size of the *PCs*' names, their exposure here is out of the question. Fortunately, it is not necessary in order to discuss them.

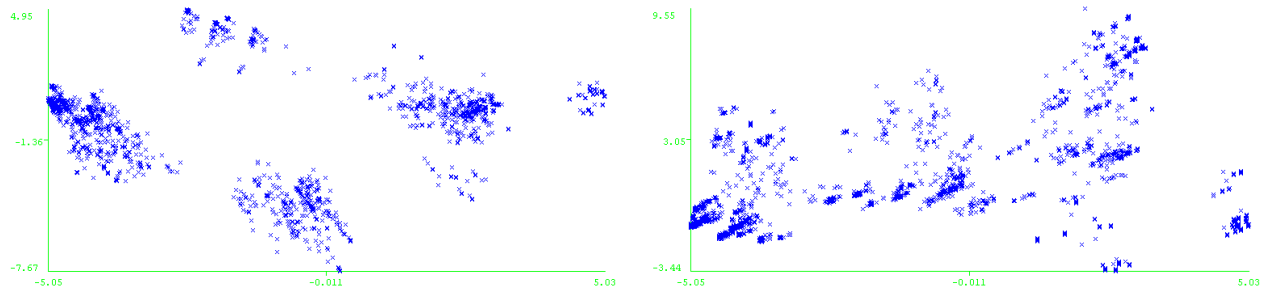
As it turns out, there are no outstanding features. There are, of course, features with larger coefficients associated to them than others, but adjacent coefficients are close in value, and the largest ones range from 0.228 (first *PC*) to 0.295 (second *PC*). From these, they gradually decrease in value, until they reach zero. Interestingly enough, one of the features that would seem more promising achieved low coefficients in all three *PCs*. This feature is the *log_client_time*, and its lowest coefficient value is 0.013, verified in both the first and second *PCs*. This may be due to the fact that this feature varies *too much* in comparison to the rest. Another interesting aspect is that, although this user has about five different emails, the feature that represents one of them is always on the top twenty, accompanied by the plugin feature related to the use of *Microsoft Outlook* and the action type feature related to mark emails. On the other hand, another *log_email_from* feature - the one that represents the empty email, i.e., the case where the user does not send an email - and the action type features related to classifying documents and files are also easily found amidst the larger coefficient values. In the same direction, the attachment features scored very low in all three *PCs*. This might mean that the user's work is more or less distributed between sending emails and working directly with files and/or documents.

Most of the features achieve, more or less, the same coefficients throughout the three *PCs*. There is no apparent "winning" feature, except perhaps from the confidentiality order features, being that at least one of them is always ranked as one of the top five features with the largest coefficients. Moreover, being it a current or a previous confidentiality order, a confidentiality order related to public information is present in the top five features of all three *PCs*. This can mean, for instance, that the user reclassifies public data into one of the classification levels above with some frequency.

The second *PCA* run was on dataset *B*. The final number of *PC*s in this case is 1128. Figure 4.3 documents two of the outputted patterns. As before, both of the charts were created by plotting the top three most important *PCs*: (a) was obtained by plotting *PC* one against *PC* three, and (b) by plotting *PC* one against *PC* two.

As it can be expected from the number of resulting *PCs*, the three first ones do not represent a large amount of the total variance - only 1.74%, to be precise. Regardless, they do reveal some clues to which features may matter the most.

Again, there are no outstanding features. The largest coefficients range from 0.228 (first *PC*)



(a) Example pattern of dataset *B*, still similar to those of dataset *A*.

(b) A second example.

FIGURE 4.3: *PCA* clusters from dataset *B*. As in dataset *A*, the same degeneration tendency is evident throughout the plots.

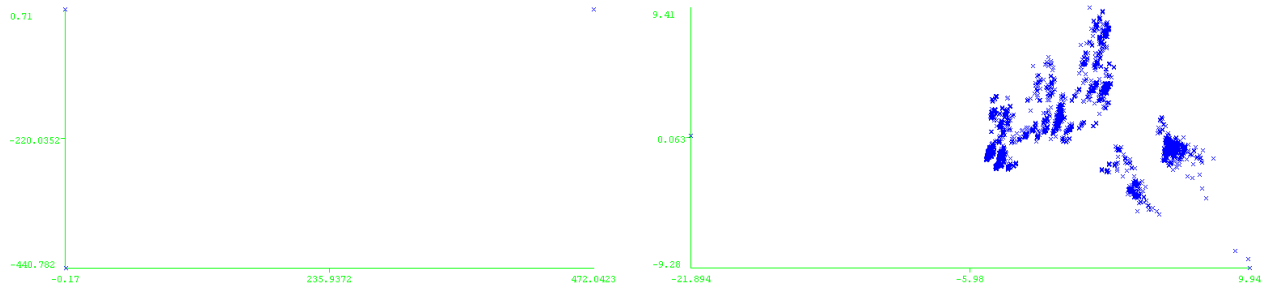
to 0.302 (second *PC*). The overall behaviour for the coefficients is similar to that obtained from dataset *A*. On the second and third *PCs*, the zero valued coefficients are all associated with file paths. On the first *PC*, however, all of the zero valued coefficients are also attached to file path features, except for one, which is attached to the client time feature. Even in the second and third *PCs*, this feature scored coefficients close to zero - 0.032 and 0.01, respectively.

Going back to the file path features, it is also visible on all three *PCs* that the vast majority of those who are not attached to a zero valued coefficient, are attached to the coefficients closer to zero. Only when these features are extremely close to their limit (in an ascending order, coefficient wise), do the rest of the features start to intermingle with them. This happens near the largest coefficient values, where the remaining features lie. Note that there are some exceptional cases though, as some file path features are among the features with the largest coefficients. These represent in particular the empty path, regarding both current and previous file paths. In fact, the largest coefficient of the second *PC* is the empty previous file path. These features can be understood as a clue to the user's behaviour, that complements the clues already uncovered with dataset *A*: it means that the majority of his classification operations are either performed over new documents created by him, or over not yet classified documents, and also that he sends a considerable amount of emails.

Following in the previous dataset's footsteps, the remaining features retain similar coefficients over the three *PCs*. Again, the confidentiality order features stand out, with at least one of them always ranked as one of the top four features with the largest coefficient. Other than this, the only thing that apparently remains constant is the ranking of all these features related to the majority of the file path features.

Taking the nature of the features and the previously seen classification results, it was expected that the path features would introduce a considerate amount of noise into the plots. As it turns out, this was not the case, and the axis limiting values of the plots from both datasets A and B are close to each other. Apart from this, the plotting results are comparable. This might indicate that the file path features possess some yet unseen potential.

While running *PCA* on dataset C, the final number of *PCs* was 1486. The resulting pattern visualisation is documented in figure 4.4.



(a) Example pattern of dataset C. The observations are barely visible. Also, the data range is much larger when in comparison to the previously seen charts.

(b) A second example of the same dataset, more similar to those of datasets A and B.

FIGURE 4.4: Examples of dataset C's patterns. As in the other two datasets, the same degeneration behaviour applies.

Chart 4.4(a) is the result of plotting the first and second *PCs*. These considered some email addresses to be the most important features. Curiously, most of these emails belong to the *cc* recipients, and the coefficient values are the same for all of them until "non-email" features appear - 0.16 and 0.171 for the first and second *PCs*, respectively.

On the other hand, chart 4.4(b) resulted by plotting the first two *PCs* that did not consider an email address to be the most relevant feature: the fourth and the fifth *PCs*. Both of them are similar to the *PCs* that were witnessed with dataset B. In fact, the described behaviour for dataset B's *PCs* still partially holds for this dataset's *PCs* - path and client time features' coeficients are close to zero, if not zero, and none of the other features clearly distinguishes itself from the rest, with the exception of the *PCs* where email addresses are the leading features. Notwithstanding, the addresses that are not at the top follow the same pattern as the path features: either with coefficients close to zero, or with zero valued coefficients. Also, more of the "non email" features (file paths excluded) are found with zero valued coefficients, which was not verified in the *PCs* from the first dataset.

As it was already expected, the charts above are quite revealing regarding the effect that the email addresses bring to the dataset. The majority of the email addresses are not used more than once - which is exactly the case of the *cc emails* that "won" the first places in the first *PC*, for instance. Also, the full set of emails correspond to approximately thirty-six percent of the whole dataset. Belonging to a significant portion of a dataset while representing such sparse occurrences is a common recipe for confusing learning algorithms. Indeed, the noise induced by the email addresses can lead to either under and over training.

In all three *PCA* runs, the correlation between features was fairly poor. Except from the obvious correlations (for instance, the *mark email* action type feature is always heavily correlated with the *Microsoft Outlook plugin* feature), most of the features have shown low correlation values between each other.

Regarding the results, the only action that will be taken will be towards the recipient email addresses. As it became clear, their presence in the dataset is a prelude to a noisy and erroneous classifier. In spite of this, simply discarding the emails is a waste of possibly useful data. As such, a more conservative solution will be considered when testing the framework. This solution has to do with the division of the recipient emails into their respective local and domain parts. The local part is discarded, while the domain is retained as a feature. Of course, this way it is not possible to distinguish, for example, if a user is sending a hundred emails to a hundred different *Gmail* addresses, or a hundred emails to the same *Gmail* address. Still, it is better than blinding the classifier with noise or not having any recipient email address information at all.

Finally, and to conclude this section, the features produced by *PCA* or, in other words, the principal components, are often used in place of the original features. In this investigation's scope, using the principal components would not be a wise decision, as the generated set of features might, in fact, be a subset of a much larger set of features. This is especially true when dealing with emails, as portrayed by the last set of plots. Anyhow, both feature extraction processes were essential to create an deeper, albeit still imperfect, understanding of the data. The next step is then to dive into the development of the *ADS*.

4.3 The *ADS* Framework

While developing the *ADS*, a variety of different obstacles, regarding different problems and modules, arose. These will all be exposed and discussed in section 4.3.1. Afterwards, section 4.3.2 will describe the *ADS* first test results, as well as some improved results by tweaking the learning algorithm's parameters. Ultimately, section 4.3.3 will provide for a moment of pause and reflection, related to the quality of the dataset so far, in the face of the last test results obtained.

4.3.1 The Implementation and its Initial Challenges

The development of the *ADS* framework started with the *Reader* module. The base code for establishing the database connections was borrowed from *RightsWATCH*. Also, the design of all the modules follow the same design patterns and structure used in *RightsWATCH*, both to provide for a better organisation and maintenance of the code, and to ease the future integration of the *ADS* into the software.

Several decisions regarding the final format of the features had to be taken while developing the *Reader* module. The most notable one is that most of the categorical ones are, once processed by this module, transformed into binary features. Since the detection algorithms that were tested deal only with numeric data, the options here were to either use one-hot encoding or to translate every categorical value to a numeric equivalent, for example, "1" for the first value encountered, "2" for the second, and so forth. The latter, although easier to implement, would induce a numeric order into something that is naturally unordered, which is incorrect and known to confuse detection algorithms. For this reason, one-hot encoding was the chosen method. As a consequence, the dimensionality of the dataset will increase, which consequentially increases the complexity and workload of the detection algorithm.

The only exception to the one-hot approach was the categorical feature *log_client_time*. The reason for this is clear: every day, hour, minute or second that passes imply the creation of a new, unique entry. Consequently, the feature would literally be translated into noise. Still,

it is unusable on its raw format, and therefore its processing by the *Reader* module is imperative. The feature can be divided into two distinct features: date and time. The date itself is too variant, but it can be used to discover the day of the week - this is useful to know, for instance, the user's working pattern throughout the week. The time, however largely variant if minutes and seconds are accounted for, can be reduced to only twenty-four values, if only the hours are considered. These options for both the date and the time are exactly what the code does: it creates these two new features, *day_of_the_week* and *time_of_the_day*, that replace *log_client_time*. Both features are numeric, being that *day_of_the_week* goes from zero (Sunday) to six (Saturday). It is important to note that *day_of_the_week* was only added later and, by that time, *time_of_the_day* was also modified[1].

Most machine learning algorithms are quite sensitive to data ranges, and *SVM* is no different. Given this and the use of one-hot encoding for categorical features, all the remaining features should be normalised between zero and one. Of course, the binary features are in this format already, but the numeric features are not, which is why they are normalised to the $[0, 1]$ interval, once the framework processes them.

At last, two modifications regarding the name of the user had to be implemented. The first one was to actually not use the *log_user* field. This was a necessary step, considering the prime objective of the framework: the anomaly detection system has to be capable of identifying the user's behaviour through the logs, and if the user's name is part of the feature set, the algorithm might eventually fall into the mistake of considering every log related to a certain user as legitimate, since the user name will never change. The exclusion of the feature is also ideal for privacy concerns. The second modification has to do with the same reasons of the first one but it happens at runtime, and it is the removal of the user's name from every directory path, whenever it is present. For instance, for a path such as "Users\John\Sales Report.docx", the final result is just "Users\Sales Report.docx".

These were all the transformations implemented on the dataset by the framework's code. Note that none of these changes directly affect the contents of the database tables.

After the preprocessing, the *Reader* module has to build the datasets. Note that the term "datasets" here means both training and test sets. This building process has some details that are very particular to the problem, especially regarding the test set. This dataset is necessary

[1] These changes will be explained in full detail on section 4.4.

to infer the quality of the classifier. The prerequisite to build it though, is the same that *WEKA* demanded: both the training and testing data **must** be composed by the same features. Only this way can the testing of the classifier be considered valid.

To create both datasets, the logs that the user generated must be split into two parts. The recommended ratio was considered: seventy percent of the total data for the training set, and the remaining thirty percent for the test set.

The datasets are created in an sequential fashion - first the training set, and then the test set. This order has to be maintained, as the features of the test set will depend on the training set. In other words, a mapping of the training set's features is stored after its creation. Then it is used to confirm which features present in the test set's data are valid - if the feature exists in the mapping, it is considered a valid one; if it does not, it is excluded as if it does not belong to the user. This *closed world assumption* point of view can accurately encompass the user's essence, but it will only do so until the end of the training set - that is to say, data that is not present in the training set can exist, and if the test set happens to possess this data, this process will go awry. For instance, it is not hard (at all) to imagine the creation of the test set going wrong due to a large number of file paths that, although existing in this set, had never been seen before on the training data. This is especially unsettling given the fact that the data from different users is used to test the classifier for a specific user. Obviously, if test data from the user itself has some of its features removed, the same will happen to test sets from other users, and to a greater extent. In spite of this, the aforementioned process of dataset genesis is still used, as it encloses a very conservative definition for "anomaly" which, concerning the problem at hands and all the unknown variables involved, is the road to follow.

When the *DatasetBuilder* is done with the datasets, they are passed to the *ClassifierBuilder* module, for the actual creation and testing of the classifier. At the core of this module, *libsvm* does all the hard work of analysing the training data and creating a classifier. It also has methods to test and cross-validate data. Two very specific inner loops of *libsvm* were parallelised with *OpenMP* for performance optimisation reasons: the loop in the *get_Q* method of the *ONE_CLASS_Q* class, and the loop related to one class classification, in the *svm_predict_values* method. This way, kernel evaluations are parallelised in both testing and training steps.

Despite being a very complete *SVM* library, *libsvm* lacks simplicity of use. For this reason, the *ClassifierBuilder* module was built as its wrapper: using *libsvm*'s methods as a solid base, it

efficiently provides a variety of easy to use, intuitive methods for classifier creation and testing. Apart from this it also implements new methods for error measurement and classifier quality assessment, based on what was seen in *WEKA*: essentially, the classification results regarding false/true positives/negatives are used to build a confusion matrix for human evaluation, and then combined together to compute the **precision** and **recall** values. These performance indicators evaluate different aspects of the classifier. Precision, or *positive predictive value*, is obtained with the division of true positives by the sum of all the examples that were considered positive (i.e., true positives and false positives) and represents the accuracy of the model in correctly predicting the class. It can be though as a numerical representation of the model's *exactness*. Conversely, recall or *sensitivity* is calculated by dividing true positives by the sum of true positives with false negatives which, in simpler terms, can be seen as the model's ability to classify observations from a class as cases from that actual class. It can be understood as the classifier's *completeness*. Both values vary between zero and one - zero being the worst case, and one the best case. Low precision can be a sign of a large number of false positives, and low recall can mean that the classifier is detecting too many false negatives. As such, the ideal is to have both values as closer to one as possible. Often, there is an inverse relationship between these two variables, which means that one can increase one value at the cost of decreasing the other one. Also, it is not common for them to be used separately. In fact, they are either used in conjunction by comparison between them or merged into a single value.

The *ClassifierBuilder* does both, calculating both measures and determinining the **F1 score**, which is a combination of the two. F1 score can be interpreted as the weighted average of precision and recall, and its computation is achieved through the *harmonic mean*[2] of both values. As precision and recall, F1 score varies between zero and one, in the same terms. These measures are used due to the fact that they are one of the best resources capable of efficiently evaluating classifiers generated from skewed classes, which is commonly the case in anomaly detection (Dokas et al., 2002).

Finally, and apart from normal model selection and model testing features, the *ClassifierBuilder* is also able to gather everything into a loop that performs a grid search over the ν and γ parameters with k-fold cross validation, using them to create one-class classifiers (SVDD, and consequentially the search for the C parameter, were only introduced in section 4.5) with each

[2] F1 score $= 2 \cdot \frac{precision \cdot recall}{precision + recall}$

of the four available kernels. The results of the four classifiers are then evaluated, and afterwards stored along with the parameters used. When the grid search ends, the *ClassifierBuilder* has both the kernel and the parameters of the classifier that performed best, along with results from the performance metrics. Since this is done for each user, it is not viable to actually use this in a real life environment, as the grid search takes too much time and real world companies do not have just four users. To make matters worse, new classifiers still have to be created with the best parameters, and then tested for quality assurance. Still, this configuration will be maintained as a way to find if there are any kernels or parameter value ranges that produce acceptable results most of the times, so that they can be used as starting points.

4.3.2 Initial Results

While the *ADS* framework was being developed, the users kept doing their usual work. When the framework became finally ready to output classification results, the available data logs rise from 13713 to 25777 - 10098, 5077, 6661 and 3761 logs belonging to user 1, 2, 3 and 4, respectively. Of course, the new tests will make use of this increase in the available data. One might argue if all the testing and feature extraction and selection processes that were previously done should not be executed again, given the increase of approximately 187% in the total available data. Regardless, this option was discarded for two simple reasons. The first reason was the lack of time - the dataset creation phase had already consumed most of the time available for the investigation. The second reason was the belief that having more data affects only the classification results. Indeed, as the four test subjects kept performing the same type of work activity, the nature of the data they generated remains the same as before, which would consequentially produce, for instance, similar *PCA* results.

For the testing methodology, it was decided to follow, more or less, the procedure suggested by *libsvm*'s authors: after converting the data to the *libsvm* format and properly scaling it, the authors encourage the use of the *RBF* kernel, before any other. In this investigation, not only the *RBF* kernel was used, but the other three kernels as well - every problem is a different problem, and for this reason it is not possible to safely assume that the *RBF* kernel, although

the most famous kernel for usually producing the best results, will also have the best performance in this case. In fact, there are some cases, such as when the number of features is very high, where a linear kernel will perform better than the *RBF* kernel. After the initial tests, the next step is to find the best parameters C (or ν, in this case) and γ via cross-validation. Finally, use the best parameters to train new classifiers and test them.

The models for the different kernels were trained on seventy percent of each user's full data. This way, thirty percent was left for testing, as well as all the data from the other users. Two very similar training datasets, built from the same logs, were used for each user, which means that in total every user has two classifiers. The only difference between the two datasets is a subset of features - one of them does not have the features that are completely unique to each user: hostname and user email addresses. This was done to evaluate the weight that such features will have on the classifier's decisions. Note that in a regular situation, these features would be vital in the dataset. In this case, as most of the test sets were composed by data from other users, it was expected that this set of features would introduce a strong bias on the results. For similar reasons, the username feature was completely excluded from both datasets as well.

Let dataset A be the full featured one, and dataset B the one without the unique features. For this testing phase, the default values for the parameters - 0.5 to ν and $\frac{1}{features}$ to γ - were used. For the polynomial kernel, the degree was the default value of 3 (and it will remain this way for the remaining tests). Due to space reasons, the classification results for both datasets are kept in section B.1.1 of appendix B. The values for precision, recall and F1 scores were computed only for the attacks with all the available test data and are available on the tables below. These values were obtained from the confusion matrices documented in section C.1.1 of appendix C. For visualisation purposes, every decimal value was rounded from six to two decimal places.

	User 1 Metrics			User 2 Metrics			User 3 Metrics			User 4 Metrics		
Kernels	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
Linear	1	0.65	0.79	1	0.11	0.19	1	0.53	0.69	1	0.58	0.73
RBF	0.16	0.68	0.26	0.10	0.25	0.14	1	0.52	0.68	0.24	0.74	0.37
Polynomial	0.26	0.84	0.39	1	0.09	0.17	1	0.55	0.71	1	0.25	0.40
Sigmoid	1	0.65	0.78	1	0.11	0.20	1	0.46	0.63	1	0.58	0.74

TABLE 4.5: Performance metrics for Dataset A.

	User 1 Metrics			User 2 Metrics			User 3 Metrics			User 4 Metrics		
Kernels	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
Linear	0.14	0.65	0.24	0.08	0.57	0.14	0.22	0.53	0.31	0.06	0.55	0.11
RBF	0.14	0.78	0.24	0.07	0.71	0.14	0.18	0.41	0.25	0.06	0.71	0.10
Polynomial	0.15	0.76	0.25	0.07	0.49	0.12	0.22	0.51	0.30	0.05	0.21	0.08
Sigmoid	0.14	0.65	0.24	0.08	0.53	0.13	0.22	0.47	0.30	0.06	0.60	0.11

TABLE 4.6: Performance metrics for Dataset *B*.

Let the focus be on dataset *A* results, for now. The number of features vary between users - 1781, 819, 377 and 761 for users 1, 2, 3 and 4, respectively. As it was expected, most of the classifiers that were trained on full featured datasets were heavily influenced by the unique features. Otherwise, it would be virtually impossible for any classifier to achieve a classification accuracy of 100%, as it happens in so many cases. Regardless, it seems that most of the classifiers whilst easily recognising other users, struggle to recognise the user itself. A particularly noteworthy case of this is user 2. As a clear case of the closed world assumption point of view discussed in section 4.3.1, this user scored the lowest when tested against itself no matter the kernel used and, as a consequence, had the lowest F1 score values too, given the low recall results. Also, notice that regardless of the unrealistically high classification values that this user achieved with the test dataset that comprises every user, the F1 score values resist this tendency and provide for a more grounded analysis, which ends up proving the advantage of using such a metric. As for the kernels themselves, the *RBF* kernel achieved the overall worst results, while both linear and sigmoid kernels appear to be more suited to deal with this kind of data, and/or this kind of problem.

The results are slightly different for dataset *B*. With a minimal decrease in features - 1775, 816, 374 and 758 for users 1, 2, 3 and 4, respectively -, the importance of the missing features becomes even clearer, as the overall results are much lower in quality. With this dataset, the classifiers had trouble both in distinguishing between users and in recognising the target user. In addition, these results also prove that the default *SVM* parameters are, in this case, far from optimal. Finally, the kernels maintained similar results between them, with the linear kernel slightly outstanding itself from the rest, always within the highest F1 score values for all the users. Note that in contrast with dataset *A*, the recall values for dataset *B* were always higher than the corresponding precision - not due to an increase in recall values, but instead because of a drastic decrease in precision.

After testing the kernels, the next proposed step is to search for the best (ν, γ) combination, through cross-validation. To follow this step, the same seventy-thirty division was made for

each user. The training sets were used in a 10-fold cross validation for a grid search on twelve values for ν and fifteen values for γ (ν goes from 2^{-25} to 1, and γ from 2^{-25} to 2^5, with increments to the exponent of 2 per iteration). In the end, the best performing combination for each dataset is stored, and then used to create new models for the respective training set. The reason for doing the grid search on only seventy percent of the data, instead of the whole dataset, was that this division was already done in the initial kernel testing, and the best parameter combination can be influenced by dataset size - of course, the same might happen when subdividing the dataset in cross-validation but, in practice, the parameters obtained here are suitable for the full training set as well.

The resulting grid search parameters are available in section D.1 of appendix D. Accuracy values were rounded to one decimal place, while everything else was rounded to two decimal places. Where two decimal places weren't enough, scientific notation had to be used. Note that the values were rounded **only** in the tables, for viewing purposes. In other words, the test runs with the new parameters used their original values.

At a first glance, and judging by the accuracy values while knowing that the training sets contain only information from the user that the classifier is supposed to infer a behavioural pattern from, it seemed that these parameters would push the framework into producing overly *permissive* classifiers. This has to do with the way that one-class *SVM* works: tampering with the ν parameter has a direct impact on the number of observations that are accepted - the lower the value, the more permissive the resulting classifier will be. The new classification results and corresponding confusion matrices, accessible in sections B.1.2 and C.1.2 of appendices B and C, respectively, portray this exact situation, particularly in dataset *B* results data. Below are the performance metrics derived from the confusion matrices as before.

	User 1 Metrics			User 2 Metrics			User 3 Metrics			User 4 Metrics		
Kernels	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
Linear	0.99	0.97	0.97	1	0.2	0.33	0.99	0.99	0.99	0.99	0.99	0.99
RBF	0.17	0.98	0.29	0.04	0.54	0.08	0.17	0.97	0.28	0.06	0.99	0.11
Polynomial	1	0.95	0.98	1	0.19	0.32	0.2	0.72	0.31	1	0.97	0.98
Sigmoid	0.17	0.99	0.3	0.07	0.48	0.12	0.24	0.99	0.39	0.06	1	0.11

TABLE 4.7: Performance metrics for Dataset *A* after grid search.

Dataset *A* still demonstrates the importance of the user unique features. In spite of this, the classifiers' tolerance is evident especially through the confusion matrices. Also outstanding

	User 1 Metrics			User 2 Metrics			User 3 Metrics			User 4 Metrics		
Kernels	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
Linear	0.16	0.98	0.27	0.07	0.97	0.14	0.11	0.99	0.2	0.05	0.99	0.1
RBF	0.16	0.98	0.28	0.07	0.99	0.13	0.09	0.99	0.17	0.05	0.99	0.1
Polynomial	0.16	0.93	0.27	0.08	0.94	0.14	0.16	0.72	0.27	0.06	0.95	0.1
Sigmoid	–	0	–	0.07	1	0.13	0.09	1	0.17	0.05	1	0.1

TABLE 4.8: Performance metrics for Dataset B after grid search.

is the clear difference (in the ability to correctly classify both anomalous and legitimate instances) between the linear kernel and the remaining three. The quality of this kernel was already noticed before, but not to this extent. For every user, this kernel attained the best F1 score. In fact, the F1 scores for this kernel are, for every user, better than those reported in the results before grid search. Although, it was the only kernel to achieve such feat, as the remaining F1 scores are mostly deplorable. Finally, the recall values for this dataset were, due to the classifiers' permissiveness, substantially higher than before, as most of the user's own observations were correctly evaluated.

The results for dataset B turned out to be even worse than before. Without the unique features, and with such forgiving ν parameters, the classifiers declare a large part of the attacking users' instances as legitimate. In this dataset, apparently, no kernel outstood from the others, and even with the high recall values, the F1 scores are simply unacceptable.

4.3.3 Conclusions

At a first glance, this investigation could end here. The linear kernel achieved good results with dataset A, after the grid search. However, this does not prove that the linear classifier will do an amazing job detecting anomalies. In fact, without the unique features (dataset B), the classifier does not perform well, at all. What if the user himself leaked valuable information, from his usual machine, with his usual email? Thanks to the unique features, the classifier would likely detect the resulting log as legitimate. As such, it is not time to finish, but to stop for a while. Time to pause and think about the causes of the results obtained with dataset B. Was it the classification algorithm, or the data itself? It might have been both.

As it was stated on section 2.4.2.6, the ν parameter controls how many observations get misclassified, and how many turn into support vectors. For instance, if the ν parameter is set

to 0.1 (recall that it is bounded between 0 and 1), it is guaranteed that at most 10% of the training instances will be misclassified, and at least 10% of them will become support vectors. The problem with upper bounding the misclassified training examples is that, consequently, the size of the margin will be smaller, as the upper bound gets smaller. Also, notice that by "misclassified training examples", one is not referring to anomalous instances, but actually referring to examples that are *not classified as legitimate*, i.e., do not fall on the correct side of the hyperplane. All of this can be easily rephrased to explain the classification results both before and after the grid search: Before the grid search, ν was the default *libsvm* value of 0.5. This is a conservative value, which created classifiers incapable of correctly recognising their legitimate user by setting a high upper bound for the outlier ratio. Then, the grid search chose the parameters that allowed for the highest F1 scores. Which such low ν values, the upper bound of outliers decreased, therefore letting more examples fall on the correct side of the hyperplane. On the other side, assuming too small outlier ratios can easily allow anomalous instances to fall on the legitimate side of the hyperplane, which indeed happened. Although not exactly, what happened was similar to **overfitting**: the classifier performs amazingly well with the training set, but fails to generalise on the test set. Of course, the γ parameter also influences everything as well. This parameter defines how *far* the influence of a single training example reaches: low values mean *far* and high values mean *close*. Before the grid search, the value was dictated by the number of features, which means that it varied, depending on the dataset, between ≈ 0.0006 and ≈ 0.001. The idea of using the inverse of the number of features as the default value is simple: the more features a dataset has, the more influential each training example should be. After the grid search, most of the γ parameters achieved values that were higher than these inverses, which in turn diminished the influence of the training instances.

All of this leads to one thought: the grid search method might prove effective with two class problems, but the same might not apply to one-class classification, given the actual nature of the optimisation problem, when performing it with cross-validation. Taking this into consideration, a new grid search was performed, but now it was more like a *brute force* attack. The classifiers were tested against data from both the user and the other users and, instead of returning the parameters with the highest F1 score, the algorithm was modified to output all the data it produces for each parameter combination - accuracy, confusion matrices, precision,

recall and F1 score - to a file, so that each of the 552 classifiers (12 for the linear kernel and 180 for each of the other kernels) could be individually examined. The perusal of the data confirmed the worst: there were no cases with dataset B where any of the classifiers managed to successfully separate anomalous from legitimate records. The classifiers mostly bounced between the two extremes - either classifying most of the cases as legitimate or as anomalous. When this is not the case, the classification accuracy values just revolve around the fifty percent mark, with very low standard deviation values.

Standing before these results, it was inevitable to consider the implementation of another classification algorithm, such as a neural network. Regardless, before any action of this nature is taken, one has also to question the other side of the problem, the data itself. One of the first impressions that arose when contacting for the first time with the available data, during the data cleansing and data selection phases, was that it was possible for the examples to be too **fine grained** for the *SVM* algorithm to be able to actually create effective classifiers from them. The features are poorly correlated between them, and with the naked eye, at least, it is next to impossible to distinguish between examples, if one ignores each users' unique features. Although this high granularity level is not confirmed, the obtained results do allow to consider it as a reason for the classifiers' poor quality. As such, it seems appropriate to rethink the way the dataset is built, and how the information is used.

In summary, two very different options are being considered. The first one involves reinventing the *ADS* framework by implementing a completely new anomaly detection algorithm. The second one demands remaking the dataset by lowering the granularity of the data and therefore remodelling the final dataset that the learning algorithm uses to produce a classifier. The best option here would be to try both, but there is not enough time for that. Now, only the decision of which one to try remains. After a discussion with the *Watchful* team, it was established that the dataset option should be the one that is tested. The reason behind this choice has to do with one of the final phases meant for the framework, i.e., the integration with *RightsWATCH Monitoring Console*. Since the framework is designed to work as a decision support system, it makes sense to have the information in such a format that a human system administrator is capable of reasoning about the various features that comprise each instance. Indeed, with the data as is, this task would be as difficult and stressful as trying to extract some meaning from the logs in the source database. Hence, the next section contemplates the final transformation

that the dataset underwent.

4.4 Dataset Refactoring

The current dataset is no more than a clean, filtered version of the raw data available in the logging database. Each example is almost equivalent to its raw counterpart, lacking only the information deemed unnecessary or repeated, while possessing two new features - client time and email attachment verification - created from raw ones and concatenating information from both the logging and email tables.

In this format, each data point represents an **atomic** user action - reading a document, sending a protected email, classifying a file, for instance. In other words, this high granularity data model falls into the *point anomaly* genre, where an instance **alone** can be considered anomalous before all others. As seen until now though, lowering the granularity of the dataset is one of the suggested actions to try and increase the classifiers' success rate. For this reason, instead of using the observations individually, the framework will aggregate them into larger sets, thus producing new features that, in essence, **summarise** the feature values from every example that comprises the given aggregated set. This way, the purpose of this phase is to morph the dataset into a completely different, **coarse-grained** version of the initial data, where the anomalies will now be of the *collective* type.

This is just the general idea. The real work comes when deciding *which* new features should be developed, *how* they should be developed and *why*.

4.4.1 Feature Engineering

Feature engineering is the process of manually developing the input features that a machine learning algorithm will use to produce a classifier. Although this was already done previously to some extent when the client time and attachment verification features were developed, it does not compare itself to what was done at this point.

The first task that had to be done was to decide what *kind* of information should the features represent. Since the idea is to summarise the original information, it seems appropriate to *count the frequencies* of each original feature and/or combinations (Cartesian products) of different ones. For instance, instead of checking if a given classification mark was applied on a single log, the framework will account for the number of times it was used over n logs, as well as the number of times it appears with each previous mark. This can be applied to most features, both successful and meaningfully. However, there was a set of new features that were obtained in a different fashion.

The first set was the group of path related features. Reusing the normal paths and counting their frequencies would not output a result that different from the original one, as the majority of the paths are used once. consequently, they were divided into three main groups: *drive*, *file type* and *directory*. The first group, *drive*, encases the hard drive letters, like *C* or *D*. The *file type* group has to do with all the features obtained by cross-referencing file extensions with a modest file extension dictionary, thus obtaining the file type, such as *spreadsheet*, *image* or *pdf*. The third and final group, *directory*, scans the full path and compares it to a list of well known folders or directories that might exist on a personal computer. It then outputs the associated folder, i.e., *dropbox* or *google drive*. Apart from these three groups, combinations between different directories and different disk drives are also taken into account.

The client time feature, as it was, was eliminated given its poor results in the *PCA* evaluation. In its place, a new feature with the same name was created. This feature stores the actual time of the day - morning, night, etc - given the time that is recorded on the original client time feature. Also, a new, although already mentioned feature, called *day of the week*, was added at this point. Its job is to store the day of the week in a numerical format, from Sunday (zero) to Saturday (six), extracted from the date in the original client time feature.

The mark id features were used to infer if the used marks were public or not, by accessing *RightsWATCH* configuration database. This way, a new set of features that count the number of *encryptions*, *classifications*[3], *decryptions* and the number of times it is not possible to obtain the used mark (i.e., *unknown mark*), due to some unknown error.

The last set of features is associated with the each action's success. These features log the *success rate of each action* - read, mark, unmark, etc - for all the occurrences of each one of

[3]If a file is marked with a public mark, it is a classification. If a file is marked with an *RMS* protected mark, it is an encryption

them throughout the logs that compose the new data point. On the other hand, the *overall success* of every action together is also computed into a feature.

Joining all these new features is also another one called *total actions*. As the name suggests, it stores the number of logs that were used to build up the new example. This brings forth an extremely important issue that did not exist until now. Since the purpose now is to build new instances by aggregating the older ones (i.e., turning point anomalies into collective anomalies), the most important step right now is to decide on which criteria should different examples be fused into one. As the logs are sequential, it becomes intuitive that they should be aggregated in a sequential order as well - n sequential logs will produce one single, summarised, example. What is not intuitive at all is the way that they should be aggregated - should the framework aggregate n logs at a time? Should they be aggregated through the use of a sliding window? Or should the framework consider the timing of their creation? In other words, should the framework aggregate all the logs that were produced in a minute? An hour? A day? These are all questions that cannot be answered without actually testing the possible aggregation methods. At least, one cannot say without any doubt which of the above is the best method. With the refactoring of the dataset, there is no time left to perform any tests related to this problem. Alas, the framework will, for now, make use of the one method that seems most promising, which is the creation of the new data points through the aggregation of the logs generated in a given time frame. The sliding window method was also strongly considered, but it was discouraged by the *Watchful* team as it was used in the past as the dataset creation method for *TypeWATCH* (which has a similar problem to solve), with poor results. This does **not** mean though that this method will not be tested in the future - If this new dataset is to be tested without testing its creation methods, the time frame method just seems to be the safest (blind) bet.

4.4.2 Final Dataset

The final dataset has thirty-one main features, excluding the “Anomaly” feature that is added to tell apart legitimate from anomalous observations. Most of these main features can then be forked into n subfeatures, depending on the user. Table 4.9 contains all the main features described, with examples of subfeatures, when applicable.

The dataset is now completely numerical, instead of (mostly) binary as before, which helps in reducing the number of final features per user. Each feature now represents a quantity of a given aspect of the user's behaviour, that can easily be interpreted by a human. The new combination features are in essence the Cartesian product of every possible combination for the given set of features. The combination is done in the format *main_feature=_x_vs._y_*, where *_x_* is the feature that applies at the time when the individual log was created, and *_y_* is the feature that was previously applied, in a previous action over the protected information. These combinations bring about a few advantages: they allow for a better (human) visualisation of the data, and for a more trustworthy comparison between users, as features like the file paths, that could easily be unique to each user, are now considered in from a more coarse-grained point of view. They also help to connect previous actions with newer ones, which can be really helpful when allied to *RightsWATCH*'s *information tracking* capabilities.

4.5 Final Results

The log aggregation process was performed considering each set of logs that was generated in an hour. In other words, the set of logs generated in an hour became one collective log. The starting time was defined by the very first log. This time frame was chosen for two reasons. Firstly, it is feasible that a user generates more than one log in an hour. Secondly, it is still a small enough time frame to allow for mitigating measures in case of a data leak.

With the aggregation, the number of available observations diminished significantly. Users 1, 2, 3 and 4 now have, respectively, 1576, 1722, 469 and 1466 total observations. The number of features has also changed, as it was expected with the changes regarding the path features, not to mention the fact that the features themselves are different than before. Using the same distinction between datasets, dataset A now consists of 642, 355, 485 and 398 variables for the four users (in that order), while dataset B contains 635, 349, 482 and 394 features. Note that with this dataset, user 3's training set will have a number of features higher than the number of observations, which can consequentially reshape the final results.

The testing process was the same one used with the point anomaly dataset - before performing a search for the best parameters, the default ones were used. The seventy-thirty division between training and test sets still remains, but note that this division was established only

Collective Anomaly Dataset	
Main Features	*Description*
totalActions	Total number of actions (logs) that were used to build the collective one.
timeOfTheDay	Set of features that represent the actual part of the day when a certain log was created (e.g. timeOfTheDay=afternoon).
dayOfTheWeek	Weekday in numerical form, from zero (Sunday) to six (Saturday).
encryptions	Number of encryptions.
classifications	Number of classifications.
decryptions	Number of decryptions.
unknownMark	Number of logs where the applied mark was impossible to determine.
drive	The hard drive letter regarding the directory of the marked file (e.g. drive=C).
fileType	The type of the marked file (e.g. fileType=Workbook).
directory	Set of features related to specific directories (e.g. directory=dropbox.)
driveCombinationFrequency	Set of features that consider the both drives of the current and previous action over a file (e.g. driveCombinationFrequency=Cvs.D).
directoryCombinationFrequency	Combination of the directories where the file is and was in the last time the user interacted with it (e.g. directoryCombinationFrequency=dropboxvs.desktop).
successRate	Features that store the success for each operation type (e.g. successRate=Mark file).
overallSuccess	Sum of all the successRate values for the given example.
typeFrequency	Number of times each type of action occurs (e.g. typeFrequency=Read).
stateFrequency	Number of occurrences for each possible log state (e.g. stateFrequency=connected).
pluginFrequency	Usage count for each *RightsWATCH* plugin (e.g. pluginFrequency=RightsWATCH for Office).
markFrequency	Number of times each mark is applied (e.g. markFrequency=Do Not Disclose).
markCombinationFrequency	Number of times each mark is applied after a given mark (e.g. markCombinationFrequency=Internalvs.Public).
confOrderFrequency	Frequency of each confidentiality order throughout the individual logs (e.g. confOrderFrequency=5).
confOrderCombinationFrequency	Combination of the confidentiality order of a mark that was applied to a given file/email and the confidentiality order of the mark that it previously had (e.g. confOrderCombinationFrequency=7vs.5).
departmentFrequency	Number of times that any mark from a given department appears (e.g. departmentFrequency=Workers).
departmentCombinationFrequency	Combination between the department of the applied mark and the department of the previously applied mark (e.g. departmentCombinationFrequency=Workersvs.Finance).
companyFrequency	Number of times that any mark from a given company is applied (e.g. companyFrequency=Watchful).
companyCombinationFrequency	Combination between the company of the applied mark and the company of the previously applied mark (e.g. companyCombinationFrequency=Watchfulvs.Project X).
hostnameFrequency	Number of times that a hostname belonging to a machine owned by the user appears (e.g. hostnameFrequency=192.168.1.79).
emailFromUsageFrequency	Number of times that an email is used to send protected information (e.g. emailFromUsageFrequency=ricardo.costeira@watchfulsoftware.com).
emailAttachmentFrequency	Total number of attachments for all the individual logs that comprise the collective observation.
domainToUsageFrequency	Number of times that a certain domain is used as a “to” recipient (e.g. domainToUsageFrequency=watchfulsoftware.com).
domainCcUsageFrequency	Number of times that a certain domain is used as a “cc” recipient (e.g. domainCcUsageFrequency=criticalsoftware.com).
domainBccUsageFrequency	Number of times that a certain domain is used as a “bcc” recipient (e.g. domainBccUsageFrequency=gmail.com).

Table 4.9: Classification accuracy results for the first version of the datasets with the path features.

in the individual logs. Several hours of development would be needed in order to apply this process to the new data, as it is created dynamically. Fortunately, with only the division between the individual logs, the final division of the new data between training and test sets is roughly eighty-twenty for users 1 and 3, and seventy-five-twenty-five for users 2 and 4, which are still acceptable boundaries. Also, this will be an opportunity to test the impact of different division ratios between the data.

The classification results are depicted in section B.2.1 of appendix B. As before, only the performance metrics for the attacks with data from all the users were computed, via the matrices in section C.2.1 of appendix C. These performance metrics are presented in the tables below.

	User 1 Metrics			User 2 Metrics			User 3 Metrics			User 4 Metrics		
Kernels	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
Linear	0.99	0.45	0.62	0.86	0.15	0.26	1	0.35	0.52	0.98	0.21	0.35
RBF	0.42	0.62	0.5	0.1	0.44	0.16	0.7	0.48	0.57	0.12	0.46	0.19
Polynomial	0.99	0.22	0.36	0.92	0.11	0.19	1	0.3	0.47	0.95	0.11	0.2
Sigmoid	0.99	0.45	0.62	0.86	0.15	0.26	1	0.35	0.52	0.98	0.21	0.35

TABLE 4.10: Performance metrics for the new Dataset *A*.

	User 1 Metrics			User 2 Metrics			User 3 Metrics			User 4 Metrics		
Kernels	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
Linear	0.22	0.45	0.3	0.12	0.2	0.15	0.12	0.33	0.17	0.1	0.22	0.14
RBF	0.26	0.63	0.37	0.11	0.52	0.18	0.15	0.48	0.23	0.09	0.46	0.16
Polynomial	0.21	0.25	0.23	0.24	0.1	0.15	0.12	0.3	0.17	0.09	0.08	0.09
Sigmoid	0.22	0.45	0.3	0.12	0.2	0.15	0.12	0.33	0.17	0.1	0.22	0.14

TABLE 4.11: Performance metrics for the new Dataset *B*.

Since the default parameters were used, these results are not unexpected. Indeed, they are very similar to what happened before with the previous dataset. Dataset *A* exhibits the same high precision and low recall values, while dataset *B* displays the same drop in precision, while maintaining the recall values. Even the different kernels performed in an identical fashion. On the other hand, the variation between the F1 scores of both dataset types is not wide enough to allow for any kind of conclusion just yet.

The grid search for this dataset was slightly different from before. Given the possibly poor and/or skewed results that might be obtained when using this method with this type of problem, its execution was altered to output every single result into a file, instead of just outputting the supposedly best parameters. This way, every parameter combination can be manually examined, which takes more time, but is guaranteed to allow the selection of the best combination.

Also, instead of performing the search through cross-validation, the algorithm attacked each of the 552 classifiers with the test sets that contain information about all users. This was done so that the obtained parameters were more adequate to the validation process. Indeed, in a normal situation the parameters would be generated via cross-validation, as the data belonging to the user would be the only data used. However, it was already seen that the classifiers are able to accept logs as being legitimate, and thus the purpose now is to search for parameters that can aid the classifiers into defining the best plane possible between both legitimate and anomalous instances.

The following tables gather the performance metrics for the classification results generated by attacking each user with both his and the other users' data, by creating classifiers with the grid search parameters visible in section D.2 of appendix D. All classification accuracy results are stored in section B.2.2 of appendix B, while the matrices used to output the performance metrics are in section C.2.2 of section C.

	User 1 Metrics			User 2 Metrics			User 3 Metrics			User 4 Metrics		
Kernels	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
Linear	0.57	0.72	0.64	0.25	0.41	0.31	29	0.79	0.42	0.19	0.74	0.3
RBF	0.79	0.60	0.68	0.69	0.26	0.38	1	0.62	0.76	0.72	0.27	0.39
Polynomial	0.89	0.86	0.87	0.8	0.74	0.77	0.94	0.84	0.89	0.95	0.81	0.87
Sigmoid	0.75	0.68	0.71	0.57	0.43	0.49	0.77	0.62	0.69	0.94	0.33	0.49

TABLE 4.12: Performance metrics for the new Dataset *A* after grid search.

	User 1 Metrics			User 2 Metrics			User 3 Metrics			User 4 Metrics		
Kernels	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
Linear	0.14	0.73	0.23	0.13	0.61	0.21	0.04	0.64	0.07	0.08	0.35	0.13
RBF	0.28	0.58	0.37	0.19	0.3	0.23	0.14	0.55	0.23	0.11	0.99	0.2
Polynomial	0.18	0.64	0.28	0.19	0.73	0.3	0.25	0.17	0.02	0.12	0.77	0.21
Sigmoid	0.18	0.64	0.28	0.14	0.62	0.22	0.27	0.17	0.21	0.1	0.69	0.17

TABLE 4.13: Performance metrics for the new Dataset *B* after grid search.

Dataset *A* achieved better results, as expected. Note that these are worse than those obtained with the previous dataset. Notwithstanding, the easiness of data evaluation and understanding that this dataset brings for the system administrator in a way compensates this decrease. What was not expected at all was the clear victory of the polynomial kernel over all the others. This kernel managed to obtain good results even when the others wavered.

Recall that the only reason for working on a completely different dataset had to do with dataset *B* disappointing results. As it turns out, the new dataset *B* performance metric results are

similar to the old ones - although, from a more optimistic point of view, when looking at both the classification and confusion matrices results, these classifiers performed clearly better, with no exception. In fact, these are good news, since there are still so many possible ways to perform the log aggregation that were not investigated. In other words, one (or more than one) of these other options might prove itself to be more successful.

In this dataset, the polynomial kernel was not able to outstand itself from the others. Actually, while still among the kernels with best overall results, it is tied with the *RBF* kernel as both won the prize for the best kernel with two of the users.

A curious pattern emerged with both datasets. With only two exceptions (tables B.23 and B.27 - both curiously for the polynomial kernel, but for different datasets), users 1 and 3 always scored the top classification results. This is most certainly related to the data division between training and test sets. Recall that these two users are the ones with the eighty-twenty division - more data to train the classifier with, and less data to test it. As it is known, the more data there is, the more accurate should the classifier be. However, this same pattern is noted, although to a lesser extent, on section B.1.2.2 tables, which might suggest that there is more to it than the data division.

Alas, two different dataset configurations produced different results. Although these last results were not that bad, they are still behind the desired outcome. Even considering the fact that attacking the classifiers with data from other users is not the desired/recommended testing and/or validation procedure as it deviates from the root of the problem, it was expected that at least this last dataset would finally bring triumph. For this reason, and right before the ending of this dissertation, a last attempt at producing effective classifiers was launched. This attempt involved the use of another classification method, also related to *SVM*s - the Support Vector Data Description method.

4.6 The Last Attempt - *SVDD*

The *SVDD* method was stumbled upon while updating the *SVM* library. It is not part of the standard library, but it is maintained by the authors as an extension. Its applications are similar to those of the one-class classification *SVM*, although using a different algorithmic

logic.

This detection method was used only on the same dataset that was used in the previous section. In fact, the whole procedure was the same as the previous section's. Regardless, as this investigation went for quite some time at this point, this section aimed only to satisfy the curiosity around this new method, and therefore only results for tests with all the available data were performed.

The following tables unveil the performance metrics for the framework first run with the *SVDD* algorithm, using the default values suggested by the authors. Recall that, for this algorithm, the regularisation parameter C (with the default value of 1) is used in place of ν. All the remaining information generated with this algorithm are stored in appendix E, being that the classification results and confusion matrices related to these tables are documented in sections E.1.1 and E.1.2, respectively.

	User 1 Metrics			User 2 Metrics			User 3 Metrics			User 4 Metrics		
Kernels	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
Linear	0.1	1	0.19	0.14	0.99	0.24	0.03	0.96	0.05	0.11	0.99	0.2
RBF	0.1	1	0.19	0.14	0.99	0.24	0.03	0.96	0.05	0.11	0.99	0.2
Polynomial	0	0	–	0.14	0.99	0.24	0.02	0.99	0.05	0.09	0.02	0.05
Sigmoid	0.1	0.99	0.19	0.14	0.99	0.24	0.03	0.96	0.05	0.11	0.99	0.2

TABLE 4.14: Performance metrics for Dataset *A* with *SVDD*.

	User 1 Metrics			User 2 Metrics			User 3 Metrics			User 4 Metrics		
Kernels	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
Linear	0.1	1	0.19	0.14	0.99	0.24	0.03	0.96	0.05	0.11	0.99	0.2
RBF	0.1	1	0.19	0.14	0.99	0.24	0.03	0.96	0.05	0.11	0.99	0.2
Polynomial	0	0	–	0.14	0.99	0.24	0.02	0.99	0.05	0.18	0.02	0.04
Sigmoid	0.1	0.99	0.19	0.14	0.99	0.24	0.03	0.96	0.05	0.11	0.99	0.2

TABLE 4.15: Performance metrics for Dataset *B* with *SVDD*.

These are clearly the worst results obtained so far with the default *SVM* parameters. Be that as it may, it seems that this algorithm has a stronger resistance to the unique features, as results from both datasets were quite similar. This is actually a good omen as this resistance is ideal in order to obtain a classifier with good generalisation capabilities. Now the only question remaining is whether the optimal parameters are able to improve the classifier's accuracy or not.

The grid search process for this test followed the same brute force methodology described in the previous section. As such, each and every classification result, for every user, every kernel,

and every parameter combination was individually analysed and compared to the remaining. The best parameters were selected and organised into the tables in section E.2, along with the respective classification result and F1 score. Through the confusion matrices in section E.3.2, the following tables were built. The corresponding classification results are in section E.3.1.

	User 1 Metrics			User 2 Metrics			User 3 Metrics			User 4 Metrics		
Kernels	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
Linear	0.31	0.68	0.42	0.11	0.55	0.18	0.2	0.62	0.3	0.12	0.83	0.2
RBF	0.8	0.62	0.7	0.75	0.28	0.41	1	0.61	0.76	0.78	0.32	0.45
Polynomial	0.16	0.7	0.25	0.11	0.7	0.19	0.05	0.7	0.1	0.09	0.64	0.16
Sigmoid	0.62	0.62	0.62	0.11	0.56	0.19	0.31	0.62	0.42	0.16	1	0.27

TABLE 4.16: Performance metrics for Dataset A with *SVDD* after grid search.

	User 1 Metrics			User 2 Metrics			User 3 Metrics			User 4 Metrics		
Kernels	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
Linear	0.21	0.68	0.32	0.12	0.62	0.2	0.07	0.61	0.13	0.09	0.49	0.16
RBF	0.26	0.62	0.37	0.2	0.36	0.26	0.15	0.57	0.23	0.11	0.3	0.16
Polynomial	0.17	0.69	0.28	0.11	0.72	0.2	0.04	0.72	0.08	0.1	0.69	0.18
Sigmoid	0.22	0.68	0.33	0.1	0.36	0.14	0.09	0.54	0.15	0.08	0.34	0.13

TABLE 4.17: Performance metrics for Dataset B with *SVDD* after grid search.

Similarly to what happened before, both datasets had somewhat similar results, with dataset A classifiers sometimes underachieving when compared to those of dataset B, but most of the times surpassing them and, in some instances, by a large margin. As such, results with dataset A are worse than those of the previous section, while those for dataset B seem similar to the ones obtained in the previous section.

Both datasets had the same kernel outstanding itself from the rest. On dataset A, the *RBF* kernel was always ahead of the other three, easily achieving the best scores. On the other hand, this distinction was not so clear with dataset B, where the *RBF* kernel was even surpassed by the polynomial kernel, on the tests with user 4.

Again, the same classification evidence concerning users 1 and 3 that was perceivable in the previous section is clear in these results. Another noticeable aspect of the tests with this classification method is that, for each user, the C regularisation parameter is mostly constant throughout the different kernels, which leads to the belief that this regularisation parameter is stronger and more influential than the ν on the one-class classification method, which might be directly involved in this method's resistance to the unique variables. In other words, it is possible that the classifiers produced by *SVDD* are more stable than the ones generated

through the one-class classification *ν-SVM*.

This investigation is now finished. Both the *SVDD* and one-class *ν-SVM* methods outputted similar results, with some slight differences. These results were not that bad, but they were not that good either. The next section discusses the steps that can and/or should be taken from here.

Chapter 5

Final Conclusions and Future Work

"We shall not cease from exploration, and the end of all our exploring will be to arrive where we started and know the place for the first time." - T.S. Elliot

RightsWATCH is an immensely complex product. When important companies and organisations buy it and use it, they also invest a great deal of trust in the software's ability to keep their sensitive information safe. Indeed, *RightsWATCH* performs amazingly well with data. However, it has next to no control at all on how the company's collaborators use it, which is a problem in a time when the actual companies and organisations are realizing that the most dangerous threats come from the inside, especially with the **Bring Your Own Device** (or **BYOD**) culture. This is why *RightsWATCH* has a data monitoring console, and this is why this investigation is so relevant, possibly not only for *Watchful* but also for anyone or any entity that needs to ensure data safety, as the framework was built so that it can be relatively easy to adapt to new data sources. Apart from this, the fact that the framework reshapes the data into something that a human can visualise in order to gain a general perspective over user activity is also important.

Also important was the study conducted over feature impact and/or importance. Although no specific features could stand out from the rest, it was shown that they all have a part to play in defining the user's behaviour. Note that this does not mean in any way that no more (yet to be defined) features are needed - it is still an area to explore.

In spite of these aspects, the main conclusion is that, as the obtained results suggest, it is possible to define a user behaviour pattern through the features outputted by *RightsWATCH*,

that can be used to identify possible data leaks through deviating behaviour. Unfortunately, without any real anomaly examples from the actual users, one cannot assure that this framework would perform as expected in a real world situation. Nonetheless, it is a good start. The journey was long and strenuous, but the final results do show promise even if they are not as good as desired. Regardless, even while a large amount of work was completed, there is still so much more to do.

With the knowledge about what has been already done and how, it becomes easy to define a high level roadmap for the framework. The first step is to test all the different options for log aggregation that remain untested: tests for aggregating logs considering different time frames, considering a fixed number of logs and considering the use of a sliding window should be conducted. If the classifiers' performance do not improve, it might be advisable to start testing with different classification methods other than *SVM*s. Even if no other methods are tested though, there is actually a machine learning methodology different from the one used so far, whose testing with *SVM*s (and, if possible, implementation) is mandatory (the only reason why it was not tested already is because it would give birth to a whole new dissertation). This methodology is called **online learning**. Online or **incremental**[1] machine learning is similar to the standard "offline" machine learning, with the difference that the model is still updated even after the initial training, as new datapoints arrive. This would allow the framework to continuously improve the classifier, even if the user changed is behavioural pattern. In a very interesting article, Laskov suggests a way to extend the already known one-class and *SVDD* classification methods to this learning method (Laskov et al., 2006), but of course, there are many more implementation suggestions, considering both linear and non-linear kernels. This *Stack Exchange* link[2] also provides an interesting list of some notorious tools that support incremental *SVM*s (although not necessarily for classification with only one class).

After finally defining how should the logs be aggregated, as well as how the classifiers should be created, everything is prepared to set the next step in motion. *Watchful* has sold its services to companies with more than 100 000 users. Therefore, there is the need to engage the framework in performance/workload testing in order to make sure that when deployed to a customer, the framework would perform all its tasks in an acceptable time and would not take over the machines' resources. After running these tests and changing what needs to be changed

[1] Online learning and incremental learning are considered to be the same thing as often as they are not.
[2] http://stats.stackexchange.com/a/51989

accordingly, it will finally be time to start the integration with *RightsWATCH* itself. The integration process will be divided into two main stages: First, deal with the server-side issues of automatically fetching the logs, send them to the monitoring console and getting the false negatives (legitimate datapoints that were considered anomalous) back to update the classifier, if online learning is being used. Second, prepare the client part, i.e., the monitoring console, by creating a new tab for anomaly detection and the corresponding window.

The two final steps are not as essential as the previous ones, but they should be approached nonetheless. After the integration with *RightsWATCH*, an investigation should occur to infer if there is something else that *RightsWATCH* can capture on the end-user, to aid in the construction of his behavioural pattern. Another option that is being considered is to do this with *TypeWATCH*, i.e., use both classifications to determine the user's authenticity. The last and final step (for now), is to fetch usage logs from other sources, such as *AD RMS*.

This is what the future has in store for the *ADS* framework. And with this, this dissertation has reached its conclusion. What remains now is to, again, thank everyone who supported me, and to thank *Watchful Software* for the opportunity to work with them - a pleasure I wish to keep having for many years: "Work hard, play hard"!

Appendix A
Logging Database Schema

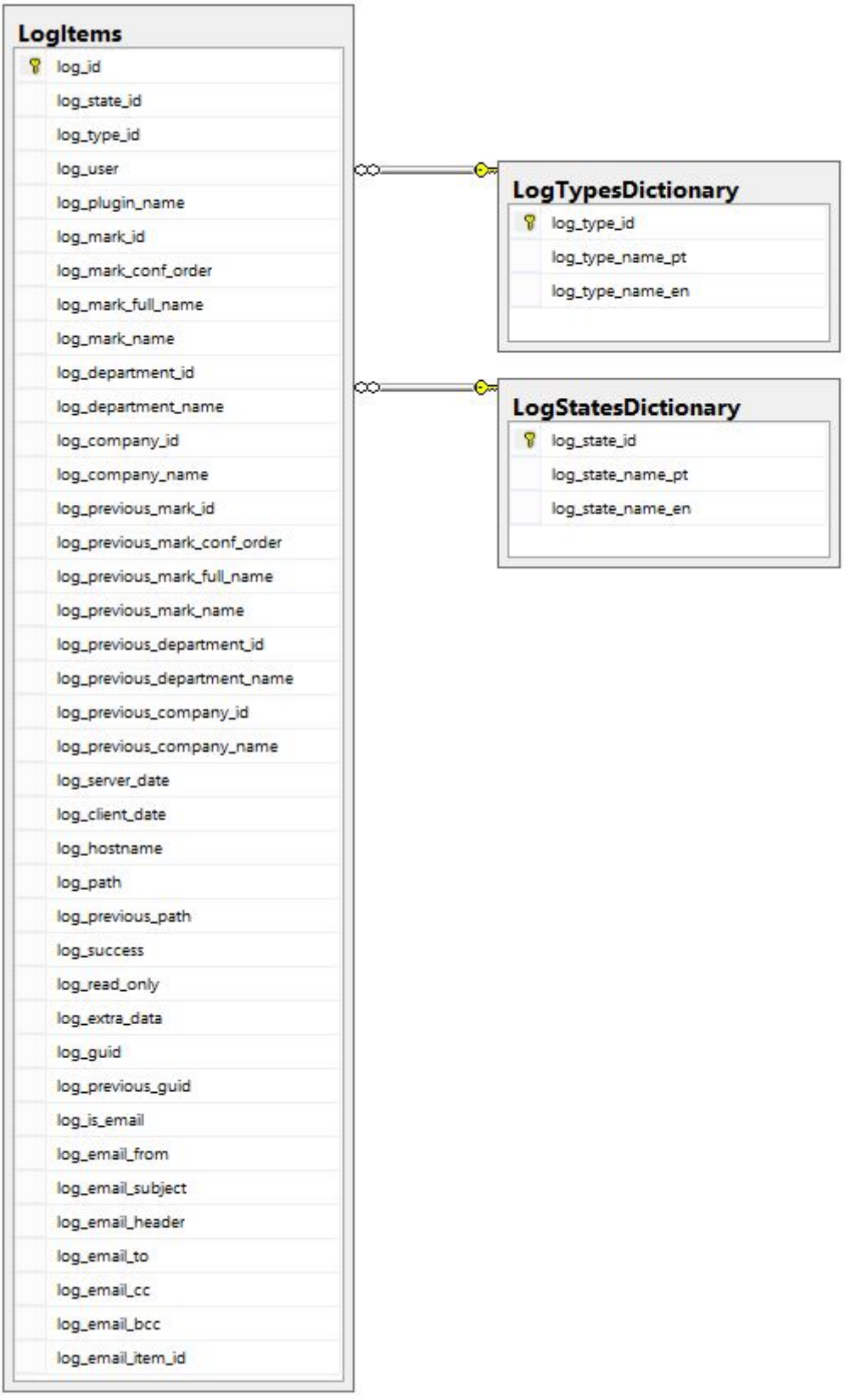


FIGURE A.1: Logging database original schema.

Appendix B
Classification Results

This appendix holds all the classification results both prior and after the grid search. The "All" column represents the values for the test sets with data from every other user plus the test data of the user whom was being tested.

B.1 Individual Logs Dataset

This section documents all the classification results obtained with the first fully functional dataset, that was composed of a clean and somewhat improved version of each log.

B.1.1 Before the Grid Search

B.1.1.1 Dataset A - Full Featured

	Classification Accuracy Results				
Anomalous / *Legitimate*	User 1	User 2	User 3	User 4	All
User 1	64.7%	100%	100%	100%	94.3%
User 2	100%	10.8%	100%	100%	93.9%
User 3	100%	100%	51.7%	100%	95.5%
User 4	100%	100%	100%	61.9%	97.8%

TABLE B.1: Classification accuracy results for dataset *A* with linear kernel.

	Classification Accuracy Results				
Anomalous / *Legitimate*	User 1	User 2	User 3	User 4	All
User 1	68.3%	27%	26.4%	41.7%	36.5%
User 2	77.2%	24.5%	93.2%	83.3%	79.5%
User 3	100%	100%	48%	100%	95.4%
User 4	83.6%	84.8%	96%	73.8%	86.9%

TABLE B.2: Classification accuracy results for dataset *A* with *RBF* kernel.

Legitimate \ Anomalous	Classification Accuracy Results				
	User 1	User 2	User 3	User 4	All
User 1	83.7%	64.3%	37.4%	66.8%	58.4%
User 2	100%	9%	100%	100%	93.8%
User 3	100%	100%	55%	100%	95.7%
User 4	100%	100%	100%	24.9%	96.1%

TABLE B.3: Classification accuracy results for dataset *A* with polynomial kernel.

Legitimate \ Anomalous	Classification Accuracy Results				
	User 1	User 2	User 3	User 4	All
User 1	64.5%	100%	100%	100%	94.3%
User 2	100%	11.2%	100%	100%	93.9%
User 3	100%	100%	46.2%	100%	94.9%
User 4	100%	100%	100%	58.3%	97.9%

TABLE B.4: Classification accuracy results for dataset *A* with sigmoid kernel.

B.1.1.2 Dataset B - No Unique Features

Legitimate \ Anomalous	Classification Accuracy Results				
	User 1	User 2	User 3	User 4	All
User 1	64%	31.8%	16%	33.3%	31.8%
User 2	63.8%	56.9%	27.4%	55.6%	50.9%
User 3	83.9%	76%	46.5%	77.29%	77.8%
User 4	64.5%	55.6%	36.1%	55.4%	53.8%

TABLE B.5: Classification accuracy results for dataset *B* with linear kernel.

Legitimate \ Anomalous	Classification Accuracy Results				
	User 1	User 2	User 3	User 4	All
User 1	77.7%	7.5%	6.7%	15.7%	20.4%
User 2	38.2%	71.2%	29.9%	37%	37.8%
User 3	82.9%	78.1%	42.8%	78.2%	76.9%
User 4	39.5%	32.3%	26.7%	75.6%	35.8%

TABLE B.6: Classification accuracy results for dataset *B* with *RBF* kernel.

Legitimate \ Anomalous	Classification Accuracy Results				
	User 1	User 2	User 3	User 4	All
User 1	76.1%	22.1%	10.2%	23.6%	26.9%
User 2	65.8%	48.9%	26.3%	59%	51.6%
User 3	84.6%	74.7%	51.1%	78.3%	77.9%
User 4	88.5%	80.8%	60.9%	21.5%	75.4%

TABLE B.7: Classification accuracy results for dataset B with polynomial kernel.

Legitimate \ Anomalous	Classification Accuracy Results				
	User 1	User 2	User 3	User 4	All
User 1	64.8%	31.8%	16%	33.3%	31.8%
User 2	65.9%	52.8%	30.5%	57.7%	52.9%
User 3	85.7%	79.1%	47.4%	79.9%	79.4%
User 4	62.7%	52.5%	30.4%	59.6%	50.9%

TABLE B.8: Classification accuracy results for dataset B with sigmoid kernel.

B.1.2 After Grid Search

B.1.2.1 Dataset A - Full Featured

Legitimate \ Anomalous	Classification Accuracy Results				
	User 1	User 2	User 3	User 4	All
User 1	97.5%	99.9%	100%	100%	99.6%
User 2	100%	19.5%	100%	100%	94.5%
User 3	99.9%	100%	98.7%	99.9%	99.9%
User 4	99.9%	100%	100%	98.9%	99.9%

TABLE B.9: Classification accuracy results for dataset A with linear kernel.

Legitimate \ Anomalous	Classification Accuracy Results				
	User 1	User 2	User 3	User 4	All
User 1	98.5%	5.9%	6.7%	12.5%	22.6%
User 2	13.4%	54.2%	10.1%	13.9%	15.3%
User 3	47.6%	47.6%	97.1%	52.8%	53.2%
User 4	13.6%	8.3%	8.1%	99.9%	15.3%

TABLE B.10: Classification accuracy results for dataset A with *RBF* kernel.

Legitimate \ Anomalous	Classification Accuracy Results				
	User 1	User 2	User 3	User 4	All
User 1	95.1%	100%	100%	100%	99.2%
User 2	100%	19.2%	100%	100%	94.5%
User 3	73.9%	64.7%	71.9%	67.4%	70.3%
User 4	100%	100%	100%	96.5%	99.8%

TABLE B.11: Classification accuracy results for dataset *A* with polynomial kernel.

Legitimate \ Anomalous	Classification Accuracy Results				
	User 1	User 2	User 3	User 4	All
User 1	99.8%	9.9%	5.7%	10.9%	23.2%
User 2	45.5%	47.9%	69.4%	43.6%	52.5%
User 3	67.5%	65.2%	99.5%	67.2%	69.9%
User 4	10.5%	7.9%	5%	100%	12.9%

TABLE B.12: Classification accuracy results for dataset *A* with sigmoid kernel.

B.1.2.2 Dataset B - No Unique Features

Legitimate \ Anomalous	Classification Accuracy Results				
	User 1	User 2	User 3	User 4	All
User 1	97.8%	1.2%	0.1%	0.7%	16.3%
User 2	18.9%	97.2%	1.3%	5.2%	16.6%
User 3	25.7%	9.1%	98.9%	14.3%	26.5%
User 4	9.2%	1.6%	0.2%	98.9%	9.5%

TABLE B.13: Classification accuracy results for dataset *B* with linear kernel.

Legitimate \ Anomalous	Classification Accuracy Results				
	User 1	User 2	User 3	User 4	All
User 1	98.4%	0.2%	0.1%	0.1%	16%
User 2	0.6%	99%	0%	0.6%	7.2%
User 3	1.9%	0.3%	98.5%	0.9%	10.5%
User 4	0.5%	0.4%	0%	99.7%	5.4%

TABLE B.14: Classification accuracy results for dataset *B* with *RBF* kernel.

Legitimate \ Anomalous	Classification Accuracy Results				
	User 1	User 2	User 3	User 4	All
User 1	93.5%	4.3%	1.7%	2.8%	17.5%
User 2	27.1%	93.7%	2.8%	9.8%	21.4%
User 3	63.7%	59.8%	72.2%	57.4%	62.4%
User 4	20.4%	6.6%	2.6%	95.4%	16%

TABLE B.15: Classification accuracy results for dataset B with polynomial kernel.

Legitimate \ Anomalous	Classification Accuracy Results				
	User 1	User 2	User 3	User 4	All
User 1	0%	100%	100%	100%	83.8%
User 2	0.1%	100%	0.03%	0.1%	6.9%
User 3	0.1%	0%	100%	0.1%	9.5%
User 4	0.1%	0.1%	0%	100%	5.2%

TABLE B.16: Classification accuracy results for dataset B with sigmoid kernel.

B.2 Collective Logs Dataset

This section gathers the classification results obtained with the second and final dataset, where each example is the result of merging all the logs that were generated in a specific time frame into one.

B.2.1 Before the Grid Search

B.2.1.1 Dataset A - Full Featured

Legitimate \ Anomalous	Classification Accuracy Results				
	User 1	User 2	User 3	User 4	All
User 1	45.22%	99.92%	100%	100%	94.38%
User 2	99.76%	15.47%	99.74%	99.36%	88.12%
User 3	100%	100%	34.83%	100%	98.45%
User 4	99.92%	99.92%	100%	21.19%	91.15%

TABLE B.17: Classification accuracy results for dataset A with linear kernel.

Legitimate \ Anomalous	Classification Accuracy Results				
	User 1	User 2	User 3	User 4	All
User 1	62.1%	95.81%	69.21%	90.71%	87.28%
User 2	32.01%	44.34%	18.16%	50.09%	38.29%
User 3	98.57%	100%	48.31%	100%	98.29%
User 4	44.61%	77.58%	31.84%	45.65%	56.14%

TABLE B.18: Classification accuracy results for dataset *A* with *RBF* kernel.

Legitimate \ Anomalous	Classification Accuracy Results				
	User 1	User 2	User 3	User 4	All
User 1	21.98%	99.92%	100%	100%	92.02%
User 2	99.92%	10.85%	100%	99.73%	87.71%
User 3	100%	100%	30.34%	100%	98.34%
User 4	99.92%	99.92%	100%	11.41%	90.1%

TABLE B.19: Classification accuracy results for dataset *A* with polynomial kernel.

Legitimate \ Anomalous	Classification Accuracy Results				
	User 1	User 2	User 3	User 4	All
User 1	45.22%	99.92%	100%	100%	94.39%
User 2	99.76%	15.47%	99.74%	99.36%	88.12%
User 3	100%	100%	34.83%	100%	98.45%
User 4	99.92%	99.92%	100%	21.19%	91.15%

TABLE B.20: Classification accuracy results for dataset *A* with sigmoid kernel.

B.2.1.2 Dataset B - No Unique Features

Legitimate \ Anomalous	Classification Accuracy Results				
	User 1	User 2	User 3	User 4	All
User 1	45.22%	90.38%	54.47%	81.33%	78.12%
User 2	78.68%	19.63%	56.84%	82.42%	69.3%
User 3	92.39%	94.1%	32.58%	95.45%	92.46%
User 4	67.75%	88.83%	57.37%	22.01%	69.69%

TABLE B.21: Classification accuracy results for dataset *B* with linear kernel.

Legitimate \ Anomalous	Classification Accuracy Results				
	User 1	User 2	User 3	User 4	All
User 1	62.74%	90.61%	45.26%	78.23%	77.77%
User 2	28.68%	52.19%	16.58%	46.54%	36.62%
User 3	85.82%	97.67%	48.31%	96.63%	92.19%
User 4	30.82%	64.1%	22.11%	45.92%	44.5%

TABLE B.22: Classification accuracy results for dataset *B* with *RBF* kernel.

Legitimate \ Anomalous	Classification Accuracy Results				
	User 1	User 2	User 3	User 4	All
User 1	24.84%	93.64%	69.74%	91.62%	82.96%
User 2	96.36%	10.39%	87.37%	95.81%	83.36%
User 3	94.06%	93.41%	30.34%	95.72%	92.8%
User 4	89.62%	92.86%	79.74%	8.42%	80.69%

TABLE B.23: Classification accuracy results for dataset *B* with polynomial kernel.

Legitimate \ Anomalous	Classification Accuracy Results				
	User 1	User 2	User 3	User 4	All
User 1	45.22%	90.38%	54.47%	81.33%	78.12%
User 2	78.61%	19.63%	56.84%	82.42%	69.27%
User 3	92.39%	94.1%	32.58%	95.45%	95.46%
User 4	67.67%	88.75%	57.37%	22.01%	69.63%

TABLE B.24: Classification accuracy results for dataset *B* with sigmoid kernel.

B.2.2 After Grid Search

B.2.2.1 Dataset A - Full Featured

Legitimate \ Anomalous	Classification Accuracy Results				
	User 1	User 2	User 3	User 4	All
User 1	71.66%	95.19%	88.16%	94.35%	91.63%
User 2	80.75%	40.88%	85%	78.87%	75.17%
User 3	93.74%	95.42%	78.65%	96.99%	94.92%
User 4	43.74%	81.07%	46.84%	74.18%	62.08%

TABLE B.25: Classification accuracy results for dataset *A* with linear kernel.

	Classification Accuracy Results				
Legitimate \ *Anomalous*	User 1	User 2	User 3	User 4	All
User 1	60.19%	99.92%	88.95%	99.45%	94.35%
User 2	98.34%	26.1%	94.74%	99.09%	88.31%
User 3	100%	100%	61.8%	100%	99.09%
User 4	97.86%	100%	97.11%	26.9%	90.69%

TABLE B.26: Classification accuracy results for dataset *A* with *RBF* kernel.

	Classification Accuracy Results				
Legitimate \ *Anomalous*	User 1	User 2	User 3	User 4	All
User 1	85.99%	98.91%	98.68%	98.63%	97.47%
User 2	96.59%	73.9%	96.84%	97.72%	97.92%
User 3	99.92%	99.69%	84.27%	100%	99.49%
User 4	98.97%	99.69%	100%	80.71%	97.33%

TABLE B.27: Classification accuracy results for dataset *A* with polynomial kernel.

	Classification Accuracy Results				
Legitimate \ *Anomalous*	User 1	User 2	User 3	User 4	All
User 1	67.52%	97.91%	92.63%	98.45%	94.35%
User 2	95.88%	42.96%	97.89%	92.9%	87.87%
User 3	99.37%	99.53%	61.8%	99.82%	98.66%
User 4	99.6%	99.84%	99.74%	32.88%	92.27%

TABLE B.28: Classification accuracy results for dataset *A* with sigmoid kernel.

B.2.2.2 Dataset B - No Unique Features

	Classification Accuracy Results				
Legitimate \ *Anomalous*	User 1	User 2	User 3	User 4	All
User 1	73.25%	52.06%	32.63%	50.36%	51.21%
User 2	30.59%	60.51%	34.21%	40.53%	38.54%
User 3	41.52%	74.01%	64.04%	59.56%	58.56%
User 4	33.12%	66.56%	38.95%	35.05%	47.07%

TABLE B.29: Classification accuracy results for dataset *B* with linear kernel.

	Classification Accuracy Results				
Anomalous / *Legitimate*	User 1	User 2	User 3	User 4	All
User 1	57.64%	92.32%	43.42%	85.25%	80.23%
User 2	82.88%	29.56%	55.53%	84.97%	73.05%
User 3	87.8%	93.79%	55.06%	94.72%	91.12%
User 4	2.14%	5.74%	0.53%	98.64%	14.13%

TABLE B.30: Classification accuracy results for dataset B with *RBF* kernel.

	Classification Accuracy Results				
Anomalous / *Legitimate*	User 1	User 2	User 3	User 4	All
User 1	63.69%	79.6%	41.32%	62.57%	67.19%
User 2	53.96%	73.21%	29.21%	52.64%	53.17%
User 3	98.26%	99.07%	16.85%	99%	96.82%
User 4	24.25%	37.01%	15%	77.17%	34.07%

TABLE B.31: Classification accuracy results for dataset B with polynomial kernel.

	Classification Accuracy Results				
Anomalous / *Legitimate*	User 1	User 2	User 3	User 4	All
User 1	64.01%	78.12%	41.32%	61.93%	66.37%
User 2	30.19%	62.36%	33.16%	46.45%	40.56%
User 3	98.34%	99.22%	16.85%	99.09%	96.92%
User 4	18.07%	26.3%	16.58%	69.02%	26.8%

TABLE B.32: Classification accuracy results for dataset B with sigmoid kernel.

Appendix C
Confusion Matrices

This appendix holds the confusion matrices for the cases where the classifier is tested against data all the users. Legitimate observations are classified as "1", while anomalous ones are classified as "−1".

C.1 Individual Logs Dataset

This section depicts the confusion matrices obtained with the first fully functional dataset regarding, of course, only the tests where the data from all the users was used to attack each user.

C.1.1 Before Grid Search

C.1.1.1 Linear Kernel

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1970	0
−1	1059	15679

TABLE C.1: Confusion matrix for User 1 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1962	11686
−1	1067	3993

TABLE C.2: Confusion matrix for User 1 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	164	0
−1	1359	20700

TABLE C.3: Confusion matrix for User 2 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	867	10247
−1	656	10453

TABLE C.4: Confusion matrix for User 2 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1052	0
−1	946	19116

TABLE C.5: Confusion matrix for User 3 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1052	3738
−1	946	15378

TABLE C.6: Confusion matrix for User 3 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	686	0
−1	496	21836

TABLE C.7: Confusion matrix for User 4 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	655	10101
−1	527	11735

TABLE C.8: Confusion matrix for User 4 - Dataset *B*.

C.1.1.2 *RBF* Kernel

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	2051	10907
−1	978	4772

TABLE C.9: Confusion matrix for User 1 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	2372	14229
−1	657	1450

TABLE C.10: Confusion matrix for User 1 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	373	3410
−1	1150	17290

TABLE C.11: Confusion matrix for User 2 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1082	13389
−1	441	7311

TABLE C.12: Confusion matrix for User 2 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1033	0
−1	965	19116

TABLE C.13: Confusion matrix for User 3 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	824	3697
−1	1174	15419

TABLE C.14: Confusion matrix for User 3 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	872	2698
−1	310	19138

TABLE C.15: Confusion matrix for User 4 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	841	14429
−1	341	7407

TABLE C.16: Confusion matrix for User 4 - Dataset *B*.

C.1.1.3 Polynomial Kernel

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	2535	7287
−1	494	8392

TABLE C.17: Confusion matrix for User 1 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	2304	12947
−1	725	2732

TABLE C.18: Confusion matrix for User 1 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	137	0
−1	1386	20700

TABLE C.19: Confusion matrix for User 2 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	744	9975
−1	779	10725

TABLE C.20: Confusion matrix for User 2 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1099	0
−1	899	19116

TABLE C.21: Confusion matrix for User 3 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1020	3692
−1	978	15424

TABLE C.22: Confusion matrix for User 3 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	294	0
−1	888	21836

TABLE C.23: Confusion matrix for User 4 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	254	4744
−1	928	17092

TABLE C.24: Confusion matrix for User 4 - Dataset *B*.

C.1.1.4 Sigmoid Kernel

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1954	0
−1	1075	15679

TABLE C.25: Confusion matrix for User 1 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1962	11686
−1	1067	3993

TABLE C.26: Confusion matrix for User 1 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	171	0
−1	1352	20700

TABLE C.27: Confusion matrix for User 2 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	804	9743
−1	719	10957

TABLE C.28: Confusion matrix for User 2 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1099	0
−1	899	19116

TABLE C.29: Confusion matrix for User 3 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1020	3296
−1	1051	15820

TABLE C.30: Confusion matrix for User 3 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	689	0
−1	493	21836

TABLE C.31: Confusion matrix for User 4 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	704	10817
−1	478	11019

TABLE C.32: Confusion matrix for User 4 - Dataset *B*.

C.1.2 After Grid Search

C.1.2.1 Linear Kernel

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	2952	1
−1	77	15678

TABLE C.33: Confusion matrix for User 1 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	2962	15584
−1	67	95

TABLE C.34: Confusion matrix for User 1 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	297	0
−1	1226	20700

TABLE C.35: Confusion matrix for User 2 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1481	18500
−1	42	2200

TABLE C.36: Confusion matrix for User 2 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1972	2
−1	26	19114

TABLE C.37: Confusion matrix for User 3 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1977	15494
−1	21	3622

TABLE C.38: Confusion matrix for User 3 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1170	2
−1	12	21834

TABLE C.39: Confusion matrix for User 4 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1169	20808
−1	13	1028

TABLE C.40: Confusion matrix for User 4 - Dataset *B*.

C.1.2.2 *RBF* Kernel

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	2983	14443
−1	46	1236

TABLE C.41: Confusion matrix for User 1 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	2981	15662
−1	48	17

TABLE C.42: Confusion matrix for User 1 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	826	18120
−1	697	2580

TABLE C.43: Confusion matrix for User 2 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1506	20615
−1	17	85

TABLE C.44: Confusion matrix for User 2 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1940	9813
−1	58	9303

TABLE C.45: Confusion matrix for User 3 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1969	18874
−1	29	242

TABLE C.46: Confusion matrix for User 3 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1181	19498
−1	1	2338

TABLE C.47: Confusion matrix for User 4 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1178	21768
−1	4	68

TABLE C.48: Confusion matrix for User 4 - Dataset *B*.

C.1.2.3 Polynomial Kernel

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	2882	0
−1	147	15679

TABLE C.49: Confusion matrix for User 1 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	2831	15232
−1	198	447

TABLE C.50: Confusion matrix for User 1 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	292	0
−1	1231	20700

TABLE C.51: Confusion matrix for User 2 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1427	17382
−1	96	3318

TABLE C.52: Confusion matrix for User 2 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1436	5713
−1	562	13403

TABLE C.53: Confusion matrix for User 3 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1442	7388
−1	556	11728

TABLE C.54: Confusion matrix for User 3 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1141	0
−1	41	21836

TABLE C.55: Confusion matrix for User 4 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1128	19275
−1	54	2561

TABLE C.56: Confusion matrix for User 4 - Dataset *B*.

C.1.2.4 Sigmoid Kernel

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	3028	14363
−1	1	1316

TABLE C.57: Confusion matrix for User 1 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	0	0
−1	3029	15679

TABLE C.58: Confusion matrix for User 1 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	738	9772
−1	785	10928

TABLE C.59: Confusion matrix for User 2 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1523	20688
−1	0	12

TABLE C.60: Confusion matrix for User 2 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1988	6340
−1	10	12776

TABLE C.61: Confusion matrix for User 3 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1998	19108
−1	1	8

TABLE C.62: Confusion matrix for User 3 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1182	20033
−1	0	1803

TABLE C.63: Confusion matrix for User 4 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	1182	21818
−1	0	18

TABLE C.64: Confusion matrix for User 4 - Dataset *B*.

C.2 Collective Logs Dataset

This section gathers the confusion matrices obtained with the second and final dataset, for the test cases pertaining data from all the users as attacking data.

C.2.1 Before Grid Search

C.2.1.1 Linear Kernel

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	142	1
−1	172	2766

TABLE C.65: Confusion matrix for User 1 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	142	502
−1	172	2265

TABLE C.66: Confusion matrix for User 1 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	67	11
−1	366	2729

TABLE C.67: Confusion matrix for User 2 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	85	626
−1	348	2114

TABLE C.68: Confusion matrix for User 2 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	31	0
−1	58	3649

TABLE C.69: Confusion matrix for User 3 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	29	222
−1	60	3427

TABLE C.70: Confusion matrix for User 3 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	78	2
−1	290	2929

TABLE C.71: Confusion matrix for User 4 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	81	713
−1	287	2218

TABLE C.72: Confusion matrix for User 4 - Dataset *B*.

C.2.1.2 *RBF* Kernel

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	195	273
−1	119	2494

TABLE C.73: Confusion matrix for User 1 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	197	568
−1	117	2199

TABLE C.74: Confusion matrix for User 1 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	192	1717
−1	241	1023

TABLE C.75: Confusion matrix for User 2 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	226	1804
−1	207	936

TABLE C.76: Confusion matrix for User 2 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	43	18
−1	46	3631

TABLE C.77: Confusion matrix for User 3 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	43	246
−1	46	3403

TABLE C.78: Confusion matrix for User 3 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	168	1247
−1	200	1684

TABLE C.79: Confusion matrix for User 4 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	169	1632
−1	199	1299

TABLE C.80: Confusion matrix for User 4 - Dataset *B*.

C.2.1.3 Polynomial Kernel

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	69	1
−1	245	2766

TABLE C.81: Confusion matrix for User 1 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	78	289
−1	236	2478

TABLE C.82: Confusion matrix for User 1 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	47	4
−1	386	2736

TABLE C.83: Confusion matrix for User 2 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	45	140
−1	388	2600

TABLE C.84: Confusion matrix for User 2 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	27	0
−1	62	3649

TABLE C.85: Confusion matrix for User 3 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	27	207
−1	62	3442

TABLE C.86: Confusion matrix for User 3 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	42	2
−1	326	2929

TABLE C.87: Confusion matrix for User 4 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	31	300
−1	337	2631

TABLE C.88: Confusion matrix for User 4 - Dataset *B*.

C.2.1.4 Sigmoid Kernel

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	142	1
−1	172	2766

TABLE C.89: Confusion matrix for User 1 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	142	502
−1	172	2265

TABLE C.90: Confusion matrix for User 1 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	67	11
−1	366	2729

TABLE C.91: Confusion matrix for User 2 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	85	627
−1	348	2113

TABLE C.92: Confusion matrix for User 2 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	31	0
−1	58	3649

TABLE C.93: Confusion matrix for User 3 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	29	222
−1	60	3427

TABLE C.94: Confusion matrix for User 3 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	78	2
−1	290	2929

TABLE C.95: Confusion matrix for User 4 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	81	715
−1	287	2216

TABLE C.96: Confusion matrix for User 4 - Dataset *B*.

C.2.2 After Grid Search

C.2.2.1 Linear Kernel

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	225	169
−1	89	2598

TABLE C.97: Confusion matrix for User 1 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	230	1419
−1	84	1348

TABLE C.98: Confusion matrix for User 1 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	177	532
−1	256	2208

TABLE C.99: Confusion matrix for User 2 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	262	1779
−1	171	961

TABLE C.100: Confusion matrix for User 2 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	70	171
−1	19	3478

TABLE C.101: Confusion matrix for User 3 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	57	1517
−1	32	2132

TABLE C.102: Confusion matrix for User 3 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	273	1156
−1	95	1775

TABLE C.103: Confusion matrix for User 4 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	129	1507
−1	239	1424

TABLE C.104: Confusion matrix for User 4 - Dataset *B*.

C.2.2.2 *RBF* Kernel

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	189	49
−1	125	2718

TABLE C.105: Confusion matrix for User 1 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	181	476
−1	133	2291

TABLE C.106: Confusion matrix for User 1 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	113	51
−1	320	2689

TABLE C.107: Confusion matrix for User 2 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	128	550
−1	305	2190

TABLE C.108: Confusion matrix for User 2 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	55	0
−1	34	3649

TABLE C.109: Confusion matrix for User 3 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	49	292
−1	40	3357

TABLE C.110: Confusion matrix for User 3 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	99	38
−1	269	2893

TABLE C.111: Confusion matrix for User 4 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	363	2828
−1	5	103

TABLE C.112: Confusion matrix for User 4 - Dataset *B*.

C.2.2.3 Polynomial Kernel

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	270	34
−1	44	2733

TABLE C.113: Confusion matrix for User 1 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	200	897
−1	114	1870

TABLE C.114: Confusion matrix for User 1 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	320	80
−1	113	2660

TABLE C.115: Confusion matrix for User 2 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	317	1370
−1	116	1370

TABLE C.116: Confusion matrix for User 2 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	75	5
−1	14	3644

TABLE C.117: Confusion matrix for User 3 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	15	45
−1	74	3604

TABLE C.118: Confusion matrix for User 3 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	297	17
−1	71	2914

TABLE C.119: Confusion matrix for User 4 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	284	2091
−1	84	840

TABLE C.120: Confusion matrix for User 4 - Dataset *B*.

C.2.2.4 Sigmoid Kernel

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	212	72
−1	102	2695

TABLE C.121: Confusion matrix for User 1 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	201	923
−1	113	1844

TABLE C.122: Confusion matrix for User 1 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	186	138
−1	247	2602

TABLE C.123: Confusion matrix for User 2 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	270	1723
−1	163	1017

TABLE C.124: Confusion matrix for User 2 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	55	16
−1	34	3633

TABLE C.125: Confusion matrix for User 3 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	15	41
−1	74	3608

TABLE C.126: Confusion matrix for User 3 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	121	8
−1	247	2923

TABLE C.127: Confusion matrix for User 4 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	254	2301
−1	114	630

TABLE C.128: Confusion matrix for User 4 - Dataset *B*.

Appendix D
Grid Search Results

This appendix documents the *SVM* parameters that achieved the best results on the grid search. Together with the parameters, the respective accuracy and F1 score that they attained for each kernel is also shown.

D.1 Individual Logs Dataset

This section provides the grid search results for the individual logs dataset, with the *SVM* parameters that were assumed as optimal for each kernel.

D.1.1 Dataset A - Full Featured

	Grid Search Best Results			
Kernels	Accuracy	F1 Score	ν	γ
Linear	99.2%	0.99	3.05×10^{-5}	
RBF	98.3%	0.99	1.95×10^{-3}	7.81×10^{-3}
Poly	96.4%	0.98	4.88×10^{-4}	0.13
Sigmoid	99.9%	0.99	4.88×10^{-4}	0.5

Table D.1: Best parameters for User 1 - dataset *A*.

	Grid Search Best Results			
Kernels	Accuracy	F1 Score	ν	γ
Linear	98.9%	0.99	4.88×10^{-4}	
RBF	98.2%	0.99	1.95×10^{-3}	1.95×10^{-3}
Poly	96.5%	0.98	1.22×10^{-4}	0.5
Sigmoid	99.9%	0.99	7.63×10^{-6}	0.5

Table D.2: Best parameters for User 2 - dataset *A*.

	Grid Search Best Results			
Kernels	Accuracy	F1 Score	ν	γ
Linear	99.2%	0.99	7.63×10^{-6}	
RBF	98.8%	0.99	1.95×10^{-3}	1.95×10^{-3}
Poly	98.3%	0.99	0.03	1.22×10^{-4}
Sigmoid	99.9%	0.99	1.22×10^{-4}	0.13

TABLE D.3: Best parameters for User 3 - dataset *A*.

	Grid Search Best Results			
Kernels	Accuracy	F1 Score	ν	γ
Linear	98.3%	0.99	1.22×10^{-4}	
RBF	97.7%	0.99	1.95×10^{-3}	1.95×10^{-3}
Poly	95.5%	0.98	1.92×10^{-7}	8
Sigmoid	99.9%	0.99	3.05×10^{-5}	0.5

TABLE D.4: Best parameters for User 4 - dataset *A*.

D.1.2 Dataset B - No Unique Features

	Grid Search Best Results			
Kernels	Accuracy	F1 Score	ν	γ
Linear	99.1%	0.99	1.22×10^{-4}	
RBF	98%	0.99	1.95×10^{-3}	1.95×10^{-3}
Poly	95.9%	0.98	1.22×10^{-4}	8
Sigmoid	100%	1	4.88×10^{-4}	8

TABLE D.5: Best parameters for User 1 - dataset *B*.

	Grid Search Best Results			
Kernels	Accuracy	F1 Score	ν	γ
Linear	98.8%	0.99	4.88×10^{-4}	
RBF	98.2%	0.99	1.95×10^{-3}	7.81×10^{-3}
Poly	96.2%	0.98	1.22×10^{-4}	8
Sigmoid	100%	1	4.88×10^{-4}	2

TABLE D.6: Best parameters for User 2 - dataset *B*.

	Grid Search Best Results			
Kernels	Accuracy	F1 Score	ν	γ
Linear	99.3%	0.99	4.88×10^{-4}	
RBF	98.8%	0.99	4.88×10^{-4}	7.81×10^{-3}
Poly	98.2%	0.99	0.03	1.22×10^{-4}
Sigmoid	99.9%	0.99	4.88×10^{-4}	2

TABLE D.7: Best parameters for User 3 - dataset *B*.

	Grid Search Best Results			
Kernels	Accuracy	F1 Score	ν	γ
Linear	98.4%	0.99	1.95×10^{-3}	
RBF	97.5%	0.99	7.81×10^{-3}	1.95×10^{-3}
Poly	95%	0.97	4.88×10^{-4}	0.13
Sigmoid	99.9%	0.99	3.05×10^{-5}	0.5

TABLE D.8: Best parameters for User 4 - dataset *B*.

D.2 Collective Logs Dataset

This section gathers the grid search results for the final dataset. This grid search was performed by attacking the user with data from both himself and all the other users, instead of cross-validation only his data.

D.2.1 Dataset A - Full Featured

	Grid Search Best Results			
Kernels	Accuracy	F1 Score	ν	γ
Linear	91.6%	0.64	4.77×10^{-7}	
RBF	94.4%	0.68	1.95×10^{-3}	0.5
Poly	97.5%	0.87	1.19×10^{-7}	0.5
Sigmoid	94.4%	0.71	1.95×10^{-3}	1.22×10^{-4}

TABLE D.9: Best parameters for User 1 - dataset *A*.

	Grid Search Best Results			
Kernels	Accuracy	F1 Score	ν	γ
Linear	91.6%	0.64	4.77×10^{-7}	
RBF	88.3%	0.38	1.95×10^{-3}	0.5
Poly	93.9%	0.77	2.98×10^{-8}	2
Sigmoid	87.9%	0.49	4.88×10^{-4}	4.88×10^{-4}

TABLE D.10: Best parameters for User 2 - dataset *A*.

	Grid Search Best Results			
Kernels	Accuracy	F1 Score	ν	γ
Linear	94.9%	0.42	0.13	
RBF	99.1%	0.76	1.95×10^{-3}	0.5
Poly	99.5%	0.89	4.88×10^{-4}	3.13×10^{-2}
Sigmoid	98.7%	0.69	0.13	3.05×10^{-5}

TABLE D.11: Best parameters for User 3 - dataset *A*.

	Grid Search Best Results			
Kernels	Accuracy	F1 Score	ν	γ
Linear	62.1%	0.3	1.95×10^{-3}	
RBF	90.7%	0.39	1.95×10^{-3}	0.5
Poly	97.3%	0.87	2.98×10^{-8}	2
Sigmoid	92.3%	0.49	0.13	7.63×10^{-6}

TABLE D.12: Best parameters for User 4 - dataset *A*.

D.2.2 Dataset B - No Unique Features

	Grid Search Best Results			
Kernels	Accuracy	F1 Score	ν	γ
Linear	51.2%	0.23	0.13	
RBF	80.2%	0.37	4.88×10^{-4}	0.5
Poly	67.2%	0.28	7.81×10^{-3}	1.95×10^{-3}
Sigmoid	66.4%	0.28	7.81×10^{-3}	2.98×10^{-8}

TABLE D.13: Best parameters for User 1 - dataset *B*.

	Grid Search Best Results			
Kernels	Accuracy	F1 Score	ν	γ
Linear	38.5%	0.21	4.77×10^{-7}	
RBF	73.1%	0.23	4.88×10^{-4}	0.5
Poly	53.2%	0.3	3.13×10^{-2}	7.81×10^{-3}
Sigmoid	40.6%	0.22	3.13×10^{-2}	4.88×10^{-4}

TABLE D.14: Best parameters for User 2 - dataset *B*.

	Grid Search Best Results			
Kernels	Accuracy	F1 Score	ν	γ
Linear	58.6%	0.07	4.77×10^{-7}	
RBF	91.1%	0.23	7.81×10^{-3}	0.5
Poly	96.8%	0.2	0.13	2.98×10^{-8}
Sigmoid	96.9%	0.21	0.13	2.98×10^{-8}

TABLE D.15: Best parameters for User 3 - dataset *B*.

	Grid Search Best Results			
Kernels	Accuracy	F1 Score	ν	γ
Linear	47.1%	0.13	4.77×10^{-7}	
RBF	14.1%	0.2	1.22×10^{-4}	3.13×10^{-2}
Poly	37.1%	0.21	1.95×10^{-3}	3.13×10^{-2}
Sigmoid	26.8%	0.17	7.81×10^{-2}	4.88×10^{-4}

TABLE D.16: Best parameters for User 4 - dataset *B*.

Appendix E
Results for the *SVDD* Method

This appendix gathers all the data generated from tests with the *SVDD* detection method. For all the tests regarding this detection method, only the datasets that use the data from all users as attacking data were used.

E.1 Before Grid Search

E.1.1 Classification Results

	Classification Accuracy Results			
Kernels / *Users*	Linear	*RBF*	Polynomial	Sigmoid
User 1	11.81%	12.82%	89.76%	11.94%
User 2	13.8%	13.84%	13.74%	18.8%
User 3	21.27%	21.7%	4.41%	21.59%
User 4	13.46%	13.76%	89.09%	14.03%

TABLE E.1: Classification accuracy results for dataset *A*.

	Classification Accuracy Results			
Kernels / *Users*	Linear	*RBF*	Polynomial	Sigmoid
User 1	11.07%	11.3%	89.74%	11.07%
User 2	13.8%	13.8%	13.74%	13.8%
User 3	13.86%	13.56%	4.79%	13.14%
User 4	11.88%	11.97%	87.91%	11.88%

TABLE E.2: Classification accuracy results for dataset *B*.

E.1.2 Confusion Matrices

E.1.2.1 Linear Kernel

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	314	2717
−1	0	50

TABLE E.3: Confusion matrix for User 1 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	314	2740
−1	0	27

TABLE E.4: Confusion matrix for User 1 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	430	2732
−1	3	8

TABLE E.5: Confusion matrix for User 2 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	431	2733
−1	2	7

TABLE E.6: Confusion matrix for User 2 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	85	2939
−1	4	710

TABLE E.7: Confusion matrix for User 3 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	85	3216
−1	4	433

TABLE E.8: Confusion matrix for User 3 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	365	2852
−1	3	79

TABLE E.9: Confusion matrix for User 4 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	365	2904
−1	3	27

TABLE E.10: Confusion matrix for User 4 - Dataset *B*.

E.1.2.2 RBF Kernel

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	314	2686
−1	0	81

TABLE E.11: Confusion matrix for User 1 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	314	2733
−1	0	34

TABLE E.12: Confusion matrix for User 1 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	431	2732
−1	2	8

TABLE E.13: Confusion matrix for User 2 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	431	2733
−1	2	7

TABLE E.14: Confusion matrix for User 2 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	85	2923
−1	4	726

TABLE E.15: Confusion matrix for User 3 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	85	3227
−1	4	422

TABLE E.16: Confusion matrix for User 3 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	364	2841
−1	4	90

TABLE E.17: Confusion matrix for User 4 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	365	2901
−1	3	30

TABLE E.18: Confusion matrix for User 4 - Dataset *B*.

E.1.2.3 Polynomial Kernel

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	0	1
−1	314	2766

TABLE E.19: Confusion matrix for User 1 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	0	2
−1	314	2765

TABLE E.20: Confusion matrix for User 1 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	432	2736
−1	1	4

TABLE E.21: Confusion matrix for User 2 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	432	2736
−1	1	4

TABLE E.22: Confusion matrix for User 2 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	88	3572
−1	1	77

TABLE E.23: Confusion matrix for User 3 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	88	3558
−1	1	91

TABLE E.24: Confusion matrix for User 3 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	9	1
−1	359	2930

TABLE E.25: Confusion matrix for User 4 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	9	40
−1	359	2891

TABLE E.26: Confusion matrix for User 4 - Dataset *B*.

E.1.2.4 Sigmoid Kernel

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	313	2712
−1	1	55

TABLE E.27: Confusion matrix for User 1 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	313	2739
−1	1	28

TABLE E.28: Confusion matrix for User 1 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	430	2732
−1	3	8

TABLE E.29: Confusion matrix for User 2 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	431	2733
−1	2	7

TABLE E.30: Confusion matrix for User 2 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	85	2927
−1	4	722

TABLE E.31: Confusion matrix for User 3 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	85	3243
−1	4	406

TABLE E.32: Confusion matrix for User 3 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	364	2832
−1	4	99

TABLE E.33: Confusion matrix for User 4 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	365	2904
−1	3	27

TABLE E.34: Confusion matrix for User 4 - Dataset *B*.

E.2 Grid Search Results

E.2.1 Dataset A - Full Featured

	Grid Search Best Results			
Kernels	Accuracy	F1 Score	C	γ
Linear	81%	0.42	1.95×10^{-3}	
RBF	94.6%	0.7	0.13	0.5
Poly	58.33%	0.25	1.95×10^{-3}	3.13×10^{-2}
Sigmoid	92.31%	0.62	1.95×10^{-3}	0.13

TABLE E.35: Best parameters for User 1 - dataset A.

	Grid Search Best Results			
Kernels	Accuracy	F1 Score	C	γ
Linear	32.6%	0.18	1.95×10^{-3}	
RBF	88.9%	0.41	1.95×10^{-3}	0.5
Poly	16.7%	0.19	1.95×10^{-3}	7.81×10^{-3}
Sigmoid	34.6%	0.19	1.95×10^{-3}	8

TABLE E.36: Best parameters for User 2 - dataset A.

	Grid Search Best Results			
Kernels	Accuracy	F1 Score	C	γ
Linear	93.3%	0.31	7.81×10^{-3}	
RBF	99.1%	0.76	0.13	0.5
Poly	65.5%	0.1	7.81×10^{-3}	7.81×10^{-3}
Sigmoid	95.9%	0.42	7.81×10^{-3}	3.13×10^{-2}

TABLE E.37: Best parameters for User 3 - dataset A.

	Grid Search Best Results			
Kernels	Accuracy	F1 Score	C	γ
Linear	27.8%	0.2	7.81×10^{-3}	
RBF	91.4%	0.45	7.81×10^{-3}	0.5
Poly	25.3%	0.16	1.95×10^{-3}	7.81×10^{-3}
Sigmoid	40.1%	0.27	1.95×10^{-3}	8

TABLE E.38: Best parameters for User 4 - dataset *A*.

E.2.2 Dataset B - No Unique Features

	Grid Search Best Results			
Kernels	Accuracy	F1 Score	C	γ
Linear	71.1%	0.32	1.95×10^{-3}	
RBF	78%	0.37	7.81×10^{-3}	0.5
Poly	63%	0.28	1.95×10^{-3}	0.5
Sigmoid	71.8%	0.33	1.95×10^{-3}	3.13×10^{-2}

TABLE E.39: Best parameters for User 1 - dataset *B*.

	Grid Search Best Results			
Kernels	Accuracy	F1 Score	C	γ
Linear	32.1%	0.2	1.95×10^{-3}	
RBF	70.9%	0.25	1.95×10^{-3}	0.5
Poly	20.3%	0.2	1.95×10^{-3}	7.81×10^{-3}
Sigmoid	41%	0.14	1.95×10^{-3}	0.13

TABLE E.40: Best parameters for User 2 - dataset *B*.

	Grid Search Best Results			
Kernels	Accuracy	F1 Score	C	γ
Linear	80.2%	0.13	7.81×10^{-3}	
RBF	91%	0.23	3.13×10^{-2}	0.5
Poly	62.7%	0.1	7.81×10^{-3}	7.81×10^{-3}
Sigmoid	85.8%	0.15	7.81×10^{-3}	4.88×10^{-4}

TABLE E.41: Best parameters for User 3 - dataset *B*.

	Grid Search Best Results			
Kernels	Accuracy	F1 Score	C	γ
Linear	41%	0.16	1.95×10^{-3}	
RBF	65.5%	0.16	0.13	0.5
Poly	29.9%	0.18	1.95×10^{-3}	7.81×10^{-3}
Sigmoid	50.9%	0.13	1.95×10^{-3}	0.13

TABLE E.42: Best parameters for User 4 - dataset *B*.

E.3 Afer Grid Search

E.3.1 Classification Results

	Classification Accuracy Results			
Kernels / *Users*	Linear	*RBF*	Polynomial	Sigmoid
User 1	81.01%	94.58%	58.33%	92.31%
User 2	32.59%	88.91%	16.74%	34.57%
User 3	93.34%	99.06%	65.46%	95.88%
User 4	27.77%	91.39%	25.34%	40.13%

TABLE E.43: Classification accuracy results for dataset *A*.

	Classification Accuracy Results			
Kernels / *Users*	Linear	*RBF*	Polynomial	Sigmoid
User 1	71.11%	77.99%	63%	71.79%
User 2	32.05%	70.94%	20.33%	41.03%
User 3	80.23%	91.01%	62.65%	85.85%
User 4	41.04%	65.47%	29.95%	50.92%

TABLE E.44: Classification accuracy results for dataset *B*.

E.3.2 Confusion Matrices

E.3.2.1 Linear Kernel

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	212	483
−1	102	2284

TABLE E.45: Confusion matrix for User 1 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	214	790
−1	100	1977

TABLE E.46: Confusion matrix for User 1 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	237	1943
−1	196	797

TABLE E.47: Confusion matrix for User 2 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	270	1993
−1	163	747

TABLE E.48: Confusion matrix for User 2 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	55	215
−1	34	3434

TABLE E.49: Confusion matrix for User 3 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	54	704
−1	35	2945

TABLE E.50: Confusion matrix for User 3 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	307	2322
−1	61	609

TABLE E.51: Confusion matrix for User 4 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	180	1757
−1	188	1174

TABLE E.52: Confusion matrix for User 4 - Dataset *B*.

E.3.2.2 RBF Kernel

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	195	48
−1	119	2719

TABLE E.53: Confusion matrix for User 1 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	196	560
−1	118	2207

TABLE E.54: Confusion matrix for User 1 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	112	41
−1	311	2699

TABLE E.55: Confusion matrix for User 2 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	157	646
−1	276	2094

TABLE E.56: Confusion matrix for User 2 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	54	0
−1	35	3649

TABLE E.57: Confusion matrix for User 3 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	51	298
−1	38	3351

TABLE E.58: Confusion matrix for User 3 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	116	32
−1	252	2899

TABLE E.59: Confusion matrix for User 4 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	109	880
−1	259	2051

TABLE E.60: Confusion matrix for User 4 - Dataset *B*.

E.3.2.3 Polynomial Kernel

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	219	1189
−1	95	1578

TABLE E.61: Confusion matrix for User 1 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	218	1044
−1	96	1723

TABLE E.62: Confusion matrix for User 1 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	302	2511
−1	131	229

TABLE E.63: Confusion matrix for User 2 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	311	2406
−1	122	334

TABLE E.64: Confusion matrix for User 2 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	62	1264
−1	27	2385

TABLE E.65: Confusion matrix for User 3 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	64	1371
−1	25	2278

TABLE E.66: Confusion matrix for User 3 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	235	2330
−1	133	601

TABLE E.67: Confusion matrix for User 4 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	255	2198
−1	113	733

TABLE E.68: Confusion matrix for User 4 - Dataset *B*.

E.3.2.4 Sigmoid Kernel

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	195	118
−1	119	2649

TABLE E.69: Confusion matrix for User 1 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	215	770
−1	99	1997

TABLE E.70: Confusion matrix for User 1 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	243	1886
−1	190	854

TABLE E.71: Confusion matrix for User 2 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	158	1596
−1	275	1144

TABLE E.72: Confusion matrix for User 2 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	55	120
−1	34	3529

TABLE E.73: Confusion matrix for User 3 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	48	488
−1	41	3161

TABLE E.74: Confusion matrix for User 3 - Dataset *B*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	368	1975
−1	0	956

TABLE E.75: Confusion matrix for User 4 - Dataset *A*.

	Confusion Matrix	
Classified as \ *Actually*	1	−1
1	126	1377
−1	242	1554

TABLE E.76: Confusion matrix for User 4 - Dataset *B*.

Bibliography

Naoki Abe, Bianca Zadrozny, and John Langford. Outlier detection by active learning. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 504–509, New York, NY, USA, 2006. ACM. ISBN 1-59593-339-5. doi: 10.1145/1150402.1150459. URL http://doi.acm.org/10.1145/1150402.1150459.

Bovas Abraham and George EP Box. Bayesian analysis of some outlier problems in time series. *Biometrika*, 66(2):229–236, 1979.

Deepak Agarwal. Detecting anomalies in cross-classified streams: a Bayesian approach. *Knowledge and Information Systems*, 11:29–44, 2007. doi: 10.1007/s10115-006-0036-4.

Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD Record*, volume 22, pages 207–216. ACM, 1993.

Debra Anderson, Thane Frivold, and Alfonso Valdes. *Next-generation intrusion detection expert system (NIDES): A summary*. SRI International, Computer Science Laboratory, 1995.

P. Anderson, James. Computer security threat monitoring and surveillance. April 1980.

Stefan Axelsson. Research in intrusion-detection systems: A survey, 1998.

Stefan Axelsson. Intrusion detection systems: A survey and taxonomy. Technical report, Department of Computer Engineering, Chalmers University of Technology, Göteborg, Sweden, March 2000.

Rebecca Bace. An introduction to intrusion detection & assessment, 1998. URL http://www.iss.net/documents/whitepapers/intrusion.pdf.

Jennifer Bayuk. Data-centric security. *Computer Fraud & Security*, 2009(3):7 – 11, 2009. ISSN 1361-3723. doi: http://dx.doi.org/10.1016/S1361-3723(09)70032-6. URL http://www.sciencedirect.com/science/article/pii/S1361372309700326.

Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

Remco R Bouckaert, Eibe Frank, Mark A Hall, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. Weka—experiences with a java open-source project. *The Journal of Machine Learning Research*, 9999:2533–2541, 2010.

Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009.

Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

Cisco. Intrusion prevention for the cisco asa 5500-x series. URL http://www.cisco.com/en/US/prod/collateral/vpndevc/ps5729/ps5713/ps4077/data_sheet_c78_459036.pdf. [Online; accessed 08-February-2014].

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3): 273–297, 1995.

Thomas M Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *Electronic Computers, IEEE Transactions on*, (3):326–334, 1965.

Dipankar Dasgupta and Fernando Nino. A comparison of negative and positive selection algorithms in novel pattern detection. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 1, pages 125–130. IEEE, 2000.

Pieter de Boer and Martin Pels. Host-based intrusion detection systems. *Amsterdam University*, 2005.

Claudio De Stefano, Carlo Sansone, and Mario Vento. To reject or not to reject: that is the question-an answer in case of neural classifiers. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 30(1):84–94, 2000.

Hervé Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8):805–822, 1999.

Dorothy E Denning. An intrusion-detection model. *Software Engineering, IEEE Transactions on*, (2):222–232, 1987.

Ozgur Depren, Murat Topallar, Emin Anarim, and M Kemal Ciliz. An intelligent intrusion detection system (ids) for anomaly and misuse detection in computer networks. *Expert systems with Applications*, 29(4):713–722, 2005.

MJ Desforges, PJ Jacob, and JE Cooper. Applications of probability density estimation to the detection of abnormal conditions in engineering. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 212(8):687–703, 1998.

Paul Dokas, Levent Ertoz, Vipin Kumar, Ar Lazarevic, Jaideep Srivastava, and Pang nig Tan. Data mining for network intrusion detection. In *Proc. NSF Workshop on Next Generation Data Mining*, page 15, 2002.

Cheri Dowell and Paul Ramstedt. The computerwatch data reduction tool. In *Proceedings of the 13th National Computer Security Conference*, pages 99–108. University of California, 1990.

Eleazar Eskin. Anomaly detection over noisy data using learned probability distributions. 2000.

Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37, 1996a.

Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. The kdd process for extracting useful knowledge from volumes of data. *Communications of the ACM*, 39(11):27–34, 1996b.

B. Feinstein and G. Matthews. Intrusion detection exchange protocol (idxp), March 2007. URL http://tools.ietf.org/html/rfc4767. [Online; accessed 17-January-2014].

João Ferreira, Henrique Santos, and Bernardo Patrão. Intrusion detection through keystroke dynamics. In *Proceedings of the 10th European Conference on Information Warfare and Security (ECIW 11)*, pages 81–90, 2011.

Stephanie Forrest, Steven A Hofmeyr, and Anil Somayaji. Computer immunology. *Communications of the ACM*, 40(10):88–96, 1997.

Thomas D Garvey and Teresa F Lunt. Model-based intrusion detection. *Proceedings of the 14th national computer security conference*, 17, 1991.

Frank E Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21, 1969.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

Simon Hawkins, Hongxing He, Graham Williams, and Rohan Baxter. Outlier detection using replicator neural networks. In *Data Warehousing and Knowledge Discovery*, pages 170–180. Springer, 2002.

Marti A. Hearst, ST Dumais, E Osman, John Platt, and Bernhard Scholkopf. Support vector machines. *Intelligent Systems and their Applications, IEEE*, 13(4):18–28, 1998.

L Todd Heberlein, Gihan V Dias, Karl N Levitt, Biswanath Mukherjee, Jeff Wood, and David Wolber. A network security monitor. In *Research in Security and Privacy, 1990. Proceedings., 1990 IEEE Computer Society Symposium on*, pages 296–304. IEEE, 1990.

Alan R Hevner, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, 28(1):75–105, 2004.

Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification, 2003.

IBM. Ibm realsecure server sensor. URL `http://www.ibm.com/ru/services/iss/pdf/realsecure-server-sensor-ss.pdf`. [Online; accessed 08-February-2014].

Koral Ilgun. Ustat: A real-time intrusion detection system for unix. Master's thesis, University of California at Santa Barbara, Santa Barbara, CA, USA, November 1992.

Paul Innella. The evolution of intrusion detection systems, November 2001. URL http://www.symantec.com/connect/articles/evolution-intrusion-detection-systems. [Online; accessed 08-February-2014].

Peter Jackson. *Introduction to expert systems*. Addison-Wesley Longman Publishing Co., Inc., 1990.

A.K. Jain, A. Ross, and S. Prabhakar. An introduction to biometric recognition. *Circuits and Systems for Video Technology, IEEE Transactions on*, 14(1):4–20, Jan 2004. ISSN 1051-8215. doi: 10.1109/TCSVT.2003.818349.

Harold S Javitz and Alfonso Valdes. The sri ides statistical anomaly detector. In *Research in Security and Privacy, 1991. Proceedings., 1991 IEEE Computer Society Symposium on*, pages 316–326. IEEE, 1991.

Kurt Jensen. *Coloured Petri nets: basic concepts, analysis methods and practical use*, volume 1. Springer, 1996.

George H John. *Enhancements to the data mining process*. PhD thesis, stanford university, 1997.

I.T. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics. Springer, 2002. ISBN 9780387954424.

Anita K Jones and Robert S Sielken. Computer system intrusion detection: A survey. *Computer Science Technical Report*, 2000.

Sandeep Kumar and Eugene H Spafford. An application of pattern matching in intrusion detection. 1994a.

Sandeep Kumar and Eugene H Spafford. A pattern matching model for misuse intrusion detection. 1994b.

Terran Lane and Carla E Brodley. An application of machine learning to anomaly detection. In *Proceedings of the 20th National Information Systems Security Conference*, volume 377. Baltimore, USA, 1997.

Pavel Laskov, Christian Gehl, Stefan Krüger, and Klaus-Robert Müller. Incremental support vector learning: Analysis, implementation and applications. *The Journal of Machine Learning Research*, 7:1909–1936, 2006.

Wenke Lee and Salvatore J Stolfo. *Data mining approaches for intrusion detection*. Defense Technical Information Center, 2000.

Teresa F Lunt, Ann Tamaru, Fred Gilham, R Jagannathan, Caveh Jalali, Peter G Neumann, Harold S Javitz, Alfonso Valdes, and Thomas D Garvey. *A real-time intrusion-detection expert system (IDES)*. SRI International, Computer Science Laboratory, 1992.

Matthew V Mahoney and Philip K Chan. Learning nonstationary models of normal network traffic for detecting novel attacks. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 376–385. ACM, 2002.

Constantine Manikopoulos and Symeon Papavassiliou. Network intrusion and fault detection: a statistical anomaly approach. *Communications Magazine, IEEE*, 40(10):76–82, 2002.

Fabian Monrose and Aviel D Rubin. Keystroke dynamics as a biometric for authentication. *Future Generation Computer Systems*, 16(4):351–359, 2000.

H. Motulsky. *Intuitive Biostatistics: A Nonmathematical Guide to Statistical Thinking*. Oxford University Press, 3^{rd} edition, 2014. ISBN 9780199730063. Appendix F.

RIPE NCC. Internet engineering task force (ietf), August 2012. URL http://www.ripe.net/internet-coordination/internet-governance/internet-technical-community/ietf. [Online; accessed 17-January-2014].

Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

Animesh Patcha and Jung-Min Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks*, 51(12):3448–3470, 2007.

Kanubhai Patel and Bharat Buddhadev. An architecture of hybrid intrusion detection system. *International Journal of Information and Network Security (IJINS)*, 2(2):197–202, 2013. URL http://iaesjournal.com/online/index.php/IJINS/article/view/1753.

A.S.K. Pathan. *The State of the Art in Intrusion Prevention and Detection*. Taylor and Francis, January 2014. ISBN 9781482203516. URL http://www.google.pt/books?id=o39cAgAAQBAJ.

C.P. Pfleeger and S.L. Pfleeger. *Security in Computing*. Prentice Hall professional technical reference. Prentice Hall PTR, 2003. ISBN 9780130355485. URL http://books.google.pt/books?id=O3VB-zspJo4C.

Ponemon Institute. 2013 cost of data breach study: Global analysis. Technical report, 2308 US 31 North, Traverse City, Michigan 49686 USA, May 2013. URL https://www4.symantec.com/mktginfo/whitepaper/053013_GL_NA_WP_Ponemon-2013-Cost-of-a-Data-Breach-Report_daiNA_cta72382.pdf.

Phillip A. Porras. Stat: A state transition analysis tool for intrusion detection. Master's thesis, University of California at Santa Barbara, Santa Barbara, CA, USA, July 1992.

Phillip A. Porras and Alfonso Valdes. Live traffic analysis of tcp/ip gateways. In *NDSS*. The Internet Society, 1998. ISBN 1-891562-01-0. URL http://www.sdl.sri.com/projects/emerald/live-traffic.html.

Leonid Portnoy, Eleazar Eskin, and Sal Stolfo. Intrusion detection with unlabeled data using clustering. In *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001*, pages 5–8, 2001.

D Romano. Data mining leading edge: Insurance & banking. In *Proceedings of Knowledge Discovery and Data Mining*, pages 144–152. Unicom, Brunel University, 1997.

Peter J Rousseeuw and Annick M Leroy. *Robust regression and outlier detection*. John Wiley & Sons, Inc, 1987.

Karen Scarfone and Peter Mell. Guide to intrusion detection and prevention systems (idps). *NIST Special Publication*, 800(2007):94, 2007.

Bernhard Schölkopf, Robert C Williamson, Alex J Smola, John Shawe-Taylor, and John C Platt. Support vector method for novelty detection. *NIPS*, 12:582–588, 1999.

Taeshik Shon and Jongsub Moon. A hybrid machine learning approach to network anomaly detection. *Information Sciences*, 177(18):3799–3821, 2007.

Stephen E Smaha. Haystack: An intrusion detection system. In *Aerospace Computer Security Applications Conference, 1988., Fourth*, pages 37–44. IEEE, 1988.

Steven R Snapp, James Brentano, Gihan V Dias, Terrance L Goan, L Todd Heberlein, Che-Lin Ho, Karl N Levitt, Biswanath Mukherjee, Stephen E Smaha, Tim Grance, et al. Dids (distributed intrusion detection system)-motivation, architecture, and an early prototype. In *Proceedings of the 14th national computer security conference*, pages 167–176. Citeseer, 1991.

Paul Spirakis, Sokratis Katsikas, Dimitris Gritzalis, Francois Allegre, John Darzentas, Claude Gigante, Dimitris Karagiannis, P Kess, Heiki Putkonen, and Thomas Spyrou. Securenet: A network-oriented intelligent intrusion prevention and detection system. *Network Security Journal*, 1(1), 1994.

Thomas Spyrou and John Darzentas. Intention modelling: approximating computer user intentions for detection and prediction of intrusions. In *SEC*, pages 319–336, 1996.

Ingo Steinwart, Don Hush, and Clint Scovel. A classification framework for anomaly detection. *J. Mach. Learn. Res.*, 6:211–232, December 2005. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1046920.1058109.

Internet Security Systems. Realsecure network sensor and gigabit network sensor frequently asked questions, a. URL http://pic.dhe.ibm.com/infocenter/sprotect/v2r8m0/topic/com.ibm.legacy.doc/rsn_faq.pdf. [Online; accessed 08-February-2014].

Internet Security Systems. Realsecure network sensor and gigabit network sensor installation guide version 7.0, b. URL http://pic.dhe.ibm.com/infocenter/sprotect/v2r8m0/topic/com.ibm.legacy.doc/RS_NetSensor_IG_7.0.pdf. [Online; accessed 08-February-2014].

David MJ Tax and Robert PW Duin. Support vector domain description. *Pattern recognition letters*, 20(11):1191–1199, 1999.

James Theiler and D. Michael Cai. Resampling approach for anomaly detection in multispectral images. In *IN PROC. SPIE*, pages 230–240, 2003.

Hank S Vaccaro and Gunar E Liepins. Detection of anomalous computer session activity. In *Security and Privacy, 1989. Proceedings., 1989 IEEE Symposium on*, pages 280–289. IEEE, 1989.

Dr. V. CH. Venkaiah, Dr. M Sreenivasa Rao, and G. Jacob Victor. Intrusion detection systems - analysis and containment of false positives alerts. *International Journal of Computer Applications*, 5(8):27–33, August 2010. Published By Foundation of Computer Science.

Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 30 Corporate Drive, Suite 400, Burlington, MA 01803, USA, 3^{rd} edition, February 2011. ISBN 9780123748560.

M. Wood and M. Erlinger. Intrusion detection message exchange requirements, March 2007. URL `http://tools.ietf.org/search/rfc4766`. [Online; accessed 17-January-2014].