NOVA IMS Information Management School

MGI

Mestrado em Gestão de Informação
Master Program in Information Management

BIG DATA ANALYTICS: A PREDICTIVE ANALYSIS APPLIED TO CYBERSECURITY IN A FINANCIAL ORGANIZATION

Pedro Filipe Martins Tourais Pereira

Project Work presented as partial requirement for obtaining the Master's degree in Information Management, with specialization in Knowledge Management and Business Intelligence

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

BIG DATA ANALYTICS: A PREDICTIVE ANALYSIS APPLIED TO CYBERSECURITY IN A FINANCIAL ORGANIZATION

by

Pedro Filipe Martins Tourais Pereira

Project Work presented as partial requirement for obtaining the Master's degree in Information Management, with a specialization in Knowledge Management and Business Intelligence

Advisor: Roberto Henriques, PhD

April 2019

To Diogo and Sílvia to whom I owe everything

ACKNOWLEDGEMENTS

I would like to thank all the support, patience and care given by my son Diogo, my wife Sílvia, my parents, my brother and the rest of my family and friends, during the development of this work, which whom without it wouldn't have been possible.

To my son Diogo, who has always seen his father working late night after work for this thesis, and my wife Sílvia who has endured it and helped me unconditionally in our daily lives and by reviewing each development of this work.

To my brother who helped providing the grounds for this work, the inspiration, the technical knowledge and limitless patience and support.

To my parents whom I haven't dedicate as much time as I should, for making them worry too many times, and for helping me in every way they could to overcome this step of my life.

To my friend Jorge Lopes for the countless hours of technical discussions, work reviews and for being supportive at all times.

To my advisor PhD Roberto Henriques for the guidance, support and constructive technical discussions to overcome the challenges of this work, and for all the knowledge shared not only during the development of this project but also during the classes and to whom I owe a lot in my technical growth.

ABSTRACT

With the generalization of the internet access, cyber attacks have registered an alarming growth in frequency and severity of damages, along with the awareness of organizations with heavy investments in cybersecurity, such as in the financial sector. This work is focused on an organization's financial service that operates on the international markets in the payment systems industry. The objective was to develop a predictive framework solution responsible for threat detection to support the security team to open investigations on intrusive server requests, over the exponentially growing log events collected by the SIEM from the Apache Web Servers for the financial service.

A Big Data framework, using Hadoop and Spark, was developed to perform classification tasks over the financial service requests, using Neural Networks, Logistic Regression, SVM, and Random Forests algorithms, while handling the training of the imbalance dataset through BEV. The main conclusions over the analysis conducted, registered the best scoring performances for the Random Forests classifier using all the preprocessed features available. Using the all the available worker nodes with a balanced configuration of the Spark executors, the most performant elapsed times for loading and preprocessing of the data were achieved using the column-oriented ORC with native format, while the row-oriented CSV format performed the best for the training of the classifiers.

KEYWORDS

Cybersecurity, Threat Detection, SIEM, Spark, Machine Learning, Financial Organization

RESUMO

Com a generalização do acesso à internet, os ciberataques registaram um crescimento alarmante em frequência e severidade de danos causados, a par da consciencialização das organizações, com elevados investimentos em cibersegurança, como no setor financeiro. Este trabalho focou-se no serviço financeiro de uma organização que opera nos mercados internacionais da indústria de sistemas de pagamento. O objetivo consistiu no desenvolvimento uma solução preditiva responsável pela detecção de ameaças, por forma a dar suporte à equipa de segurança na abertura de investigações sobre pedidos intrusivos no servidor, relativamente aos exponencialmente crescentes eventos de log coletados pelo SIEM, referentes aos Apache Web Servers, para o serviço financeiro.

Uma solução de Big Data, usando Hadoop e Spark, foi desenvolvida com o objectivo de executar tarefas de classificação sobre os pedidos do serviço financeiros, usando os algoritmos Neural Networks, Logistic Regression, SVM e Random Forests, solucionando os problemas associados ao treino de um dataset desequilibrado através de BEV. As principais conclusões sobre as análises realizadas registaram os melhores resultados de classificação usando o algoritmo Random Forests com todas as variáveis pré-processadas disponíveis. Usando todos os nós do cluster e uma configuração balanceada dos executores do Spark, os melhores tempos para carregar e pré-processar os dados foram obtidos usando o formato colunar ORC nativo, enquanto o formato CSV, orientado a linhas, apresentou os melhores tempos para o treino dos classificadores.

PALAVRAS-CHAVE

Cibersegurança, Detecção de Ameaças, SIEM, Spark, Machine Learning, Organização Financeira

INDEX

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AM	Application Master
ANN	Artificial Neural Network
API	Application Programming Interface
ATM	Automated Teller Machine
AUC	Area Under the Curve
AUC-PR	Area Under the Precision-Recall Curve
AUC-ROC	Area Under the ROC Curve
BEV	Bagging Ensemble Variation
COC	Convention on Cybercrime
CPU	Central Processing Unit
CRISP-DM	Cross Industry Standard Process for Data Mining
CSV	Comma-Separated Values
CV	Cross-validation
DDoS	Distributed Denial-of-Service
DoS	Denial-of-Service
FN	False Negative
FP	False Positive
FPR	False Positive Rate
FW	Firewall
GD	Gradient Descent
GDP	Gross Domestic Product
GI	Gini Importance
HDD	Hard Disk Drive
HDFS	Hadoop Distributed File System
HIDS	Host Intrusion Detection Systems
HTTP	Hypertext Transfer Protocol
IARPA	Intelligence Advanced Research Projects Activity
IC3	Internet Crime Complaint Center
IDS	Intrusion Detection System
IG	Information Gain
IP	Internet Protocol
ISP	Internet Service Provider

KDDCUP'99	Knowledge Discovery and Data Mining Tools Competition held in 1999
L-BFGS	Limited-memory Broyden-Fletcher-Goldfarb-Shanno optimization algorithm
LR	Logistic Regression
MB	Megabyte
MDI	Mean Decrease in Impurity
ML	Machine Learning
MLP	Multilayer Perceptron
NIDS	Network Intrusion Detection Systems
NSA	National Security Agency
ORC	Optimized Row Columnar format
OS	Operating System
OSI layer	Open Systems Interconnection model layer
PHP	Hypertext Preprocessor
POS	Point of Sale
PPV	Positive Predicted Value or Precision metric
PR curve	Precision-Recall curve
R2L	Remote to Local
RAM	Random-Access Memory
RDD	Resilient Distributed Dataset
RF	Random Forests
RM	ResourceManager
ROC	Receiver Operating Characteristic
SGD	Stochastic Gradient Descent
SIEM	Security Information and Event Management System
SMO	Sequential minimal optimization algorithm
SMOTE	Synthetic Minority Over-sampling Technique
SOC	Security Operations Center
SQL	Structured Query Language
SVM	Support Vector Machines
TB	Terabyte
TCP	Transmission Control Protocol
TN	True Negative
TNR	True Negative Rate
TP	True Positive
TPR	True Positive Rate

U2R	User to Root
URL	Uniform Resource Locator
WAF	Web Application Firewall
WEKA	Waikato Environment for Knowledge Analysis
YARN	Yet Another Resource Negotiator

1. INTRODUCTION

This project work was developed as a partial requirement for obtaining the Master's degree in Information Management with specialization in Knowledge Management and Business Intelligence from the NOVA Information Management School of the Universidade Nova de Lisboa.

Throughout the recent years, technological developments boosted by the generalization of the internet access have brought to our homes and societies new, promising and groundbreaking opportunities that are rapidly shaping the way people and organizations communicate, access and share information or even conduct commercial activities. However, with the steep growth in the usage of computer networks and personal smart devices in our everyday lives, spawned the threats and dangers of cyber attacks (Bendovschi & Al-Nemrat, 2016; Jenab & Moslehpour, 2016).

Cyber attacks are accountable for the intentional unauthorized and abusive usage of computer systems to damage and exploit data availability, integrity and confidentiality from single individuals to business organizations (Bendovschi, 2015; Jenab & Moslehpour, 2016; Rajan, Ravikumar, & Shaer, 2017). The high frequency and volume of reported damages and losses by people and organizations are estimated to be under-representative of the real impact of the attacks, either in numbers or in the intangible damages associated. In fact, one of the major concerns associated with this crimes is the inability for most companies to discover that they have been compromised or data breached internally (Center for Strategic and International Studies, 2014). By 2017, the World Economic Forum published their annual Global Risk Report, placing the technological dangers associated with large scale cyber attacks and massive incident of data fraud/theft, respectively, in 6th and 5th places in the list of risks most likely to occur in the next ten years (World Economic Forum, 2017). As a consequence, organizations and governmental entities awareness have risen significantly and a heavy investment in cybersecurity products and services have been increasing (Bendovschi, 2015; Center for Strategic and International Studies, 2014; Rajan et al., 2017). Among the most influential and sensitive business areas affected by this threat is the financial sector (Kumar, Yadav, Sharma, & Singh, 2016), where the financial organization of this dissertation work is framed.

This dissertation work is focused on a specific financial service of an organization that operates on the international markets in the payment systems industry, which allows end users and merchants to access a payment service through mobile or Point of Sale (POS) devices. Following the previously mentioned security concerns and the sensitive nature of the information handled by the company, the organization is continuously reinforcing the security of their infrastructures, services and procedures using new techniques and technologies, capable of supporting the constant monitorization, investigation and analysis over all the potential attacks that they are exposed to.

All the client requests to the organization's services are collected by Apache Web Servers, monitored through several layers of security and followed closely by Security Operations Center (SOC), a team composed of security experts. Among the technological stack of tools used, the Security Information and Event Management system (SIEM) is responsible for the collection and correlation of the system's applications and services generated log data, providing a hawk eye over the whole system ecosystem.

Currently, the implemented security system, for that service, is based on a set of a manually defined ruled-based system developed by security experts. The SOC can only open investigations on intrusive server requests attempts with high expression over the volume of information transacted. Due to the limited available human resources of experts, it is impractical to open investigations overall intrusion attempt, ending in an impossible tracking scenario of the number of false positives or negatives involved, as the process of opening investigations is not managed efficiently.

Since the launch of the financial service, the Apache Web Servers have registered an exponential growth in the volume of information from the increasing usage of the financial service. Thus, a more versatile, efficient and automatic solution urges as the manually defined rule-based system ends up being efficient but not updated fast enough for the complex and changing nature of the intrusion attempts and not accurate enough for the number of false positives flags raised.

The objective of this dissertation work is to develop a predictive framework solution responsible for threat detection (classification) to support the SOC team, over the log events collected by the SIEM from the Apache Web Servers for a particular financial service. Given the exponential volume growth of data logs from the increasing usage of the financial product, it is mandatory to enable the framework to operate under a Big Data environment. The solution is expected to accommodate the complete Data Mining (DM) processing pipeline, from data collection of the log server requests from SIEM, including data preprocessing, to the modelling of a classification algorithm. The design of the solution should also be capable of handling the issues associated with poor performant learning algorithms when subjected to a biased training from a highly imbalanced training set, typically present in intrusion detection scenarios.

From the resulting framework solution, the following analysis is expected to be produced in order to extract conclusion related to its performance: i) Analysis, evaluation and comparison of the different results produced by the proposed framework over different classification algorithms; ii) Analysis and evaluation of the impact and significance of the chosen features over the quality of the results produced for the most performant classifier; and iii) Analysis, evaluation and comparison of the storage gains using different file formats, elapsed processing times of the different stages of the framework, including the training of the of the different classification algorithms used, and of the different cluster configurations and spark-submit parameters.

The development of this work was organized in the following seven chapters. After this introduction, the second chapter, the Background, provides a context of the main agents involved in the cybercrime activities and their impact over the economic, social and political environments in which belongs the financial organization of this work. The third chapter, the Literature Review, is focused on the knowledge gathering of previous studies and approaches related with cybersecurity and Machine Learning (ML)/DM with emphasis on the intrusion detection systems (IDS), Big Data predictive solutions and SIEM-based implementations. The fourth chapter, the Methodology, is focused on the proposed solution describing all the methodologies and techniques implemented, along with all the steps produced on every stage of the pipeline process prior to the final results. This chapter exposes the issues encountered during the development of the framework and provides the reasoning and justifications behind all the decisions and technical approaches implemented to overcome them. The fifth chapter, the Results and Discussions, presents a series of analysis over the final results produced by the proposed framework solutions in order to evaluate its performance form different perspectives. Finally, the conclusions, limitations and the future works are presented on the sixth and seventh chapters.

2. BACKGROUND

In recent years the world has witnessed an increasing technological development merged with the generalization of internet access, creating, connecting and shaping the world into a new era of opportunities for everyone and anywhere. This scenario brought to our daily lives a connected reality where people and organizations interact and share data continuously. However, along with its potential uses, the same opportunities brought the threats and dangers of cyber attacks (Bendovschi & Al-Nemrat, 2016).

The generalization of technology usage, such as computer networks or personal smart devices, is being targeted by cyber attacks every day (Jenab & Moslehpour, 2016). The influence of the increasing technological developments related with internet social and commercial activities has made cybercrime to grow and diversify their approaches with new tools and techniques day by day successfully overcoming increasing complex security measures (Bendovschi, 2015).

2.1. CYBERCRIME AND CYBERSECURITY DEFINITION

Cyber attacks are perceived as intentional unauthorized and abusive exploitations of computer systems, technology-dependent enterprises and networks (Jenab & Moslehpour, 2016; Rajan et al., 2017). This nefarious activity seeks to innovatively create new tools to illegally contour access to networks, programs and data in order to damage and exploit data availability, integrity and confidentiality from single individuals to business organizations (Bendovschi, 2015). According to Rajan et al. (2017) “any crime that is committed using a computer or network, or hardware device” is considered a cybercrime.

Cyber threats can be generally divided into three main categories: i) Cyber Terror - Composed by independent organizations focused on spreading terror through the web; ii) Cybercrime - Related with the illegal activities to obtain money, confidential data or unethical hacking; and iii) Cyber War - Associated with damaging computers or information networks by one country or international organization against another (Kumar et al., 2016).

The rise and expansion of the cybercrime activity and damages lead to the natural creation of cybersecurity teams among organizations. “Cybersecurity refers to the techniques, processes and methodologies concerned with thwarting illegal or dishonest cyber-attacks in order to protect one or more computers on any type of network from any type of damage” (Mahmood & Afzal, 2014, p.130). The main objectives of cybersecurity are to safely acquire and exchange information, find vulnerabilities in applications, prevent and control information access, and protect confidential information (Mahmood & Afzal, 2014).

2.2. TYPES OF CYBER ATTACKS

With the steep growth of global internet usage, so did the variety, complexity and frequency of the cyber attack events (Mahmood & Afzal, 2014). Among the known and documented cyber attacks, seven overall groups of cyber attacks were identified as relevant for this work by the security team that supported it: i) Malware; ii) Denial-of-Service (DoS)/ Distributed Denial-of-Service (DDoS); iii) Application Layer Attacks; iv) User Attacks; v) Information Gathering; vi) Man-in-the-Middle; and vii) Phishing.

i. Malware - Is a software that is intentionally developed to perform malicious activities. The damage can be from stealing sensitive information from the victim to total destruction of the operating system (OS) including all the stored files. The most known types of malware are virus, trojan, worms and spyware (Mahmood & Afzal, 2014). As an example, a known worm called Stuxnet was a malicious software developed by the national-level intelligence agency of the United States department of defense, the National Agency Security (NSA), which destroyed 984 uranium enriching centrifuges in 15 different Iranian facilities.

ii. DoS/DDoS - A Denial of Service attack (DoS) and Distributed Denial of Service attack (DDoS) have the objective to prevent the usage of any type of service. There are many ways to perform a DoS attack. The most known are volumetric attacks (Viegas, Santin, & Oliveira, 2017), where the attacker floods the victim server with requests to be processed and starts to drop new requests in order to process the others. Likewise, the DDoS (Nikolskaya, Ivanov, Golodov, Minbaleev, & Asyaev, 2017) are very typically performed as volumetric attacks. But there are also other types of attacks that exploit the resources of the computer, such as CPU or memory that result in a DoS of the victim server (Dolev, Elovici, Kesselman, & Zilberman, 2009).

iii. Application Layer Attacks - This type of attacks depends on the application that the service is using, it can be an HTTP server, a Mobile application or even a Database. Depending on the application, there are many types of attacks that can damage the owner of the service or even the visitors. Some types of attacks are SQL injection, malicious ads, redirection to malicious websites, command injection, etc. (Mahmood & Afzal, 2014). As an example, the attacker could inject a command into the service that handles the client inputs from a mobile application and every time a client tries to validate the password, a background communication would be established to the attacker to retrieve the inputted password from the victim.

iv. User Attacks - These types of attacks are divided into two types (Viegas et al., 2017): i) Remote to Local (R2L) where an attacker gains access to a victim server by exploiting remotely an unpatched vulnerability, to later install a backdoor so that still after the service patches the vulnerability, the

attacker persists access to the service. One known vulnerability is Shellshock where an HTTP server did not validate correctly the user inputs and the attacker could inject commands directly to the victim server. The other attack is User to Root (U2R), where the attacker already has access to the victim account but has limited permissions to perform operations on the system. Therefore, he uses system vulnerabilities, such as buffer overflow, to escalate privileges to obtain administrator privileges.

v. Information Gathering - An attacker before starting an attack, starts by studying its target. This phase is critical for every attacker because they only need to have success once and with minimal impact and track possible. To do so, the attacker starts by gathering information from the target (Viegas et al., 2017). This process can be done in a passive or active way. In a passive way, it can search for information without requesting any information directly to the system, searching for the workers in social networks (such as Linkedin), or searching engines (such as Google), to later perform phishing attacks. In an active way, the attacker can validate if the service has some network ports open, try to find some private services that are not supposed to be used publicly, etc. If the attacker is already inside the network, he can passively sniff the network to find any sensitive information, or he can actively map all the network to find the weakest link to later steal sensitive data (Martorella, n.d.).

vi. Man-in-the-middle - Are attacks where, as the name implies, an attacker is in the middle of the communication between the client and the server. With these attacks, a client assumes that he is interacting directly with the intended service, but the attacker "in the middle" is eavesdropping or changing the information to their benefit (Luettmann & Bender, 2007). Typically, such an attack could be launched through vulnerabilities in the communication protocols used or even by a misconfiguration on the services.

vii. Phishing - Is a type of attack where the attacker intends to fraud the victim by presenting some type of information, normally an email, that is very similar to legitimate services and steal the credentials or install some backdoor in the computer (Mahmood & Afzal, 2014). One common example is emails from a suppose Apple company that informs the victim that someone has accessed the phone and they need to verify if you are the real user of the phone, so they ask you the password, which will be delivered to the attacker. The most dangerous type of phishing is called, Spear phishing (Jenab & Moslehpour, 2016). In this type, the attacker knows the victim and tries to take advantage of information that he is expecting, for example, a document that the victim needs to read before some meeting in the next day.

2.3. CONSEQUENCES OF CYBERCRIME

In 2011, the Department of Commerce Internet Policy Task Force of the United States of America pointed the exponential growth of cyber attacks on commerce, business and government agencies (The Department of Commerce Internet Policy Task Force, 2011). The number of cyber attacks and the estimated damage have been increasing consistently year after year until the year of 2014, which has been named as the "the year of cyber-attacks" (Bendovschi, 2015). "Some estimates suggest that, in the first quarter of this year [2011], security experts were seeing almost 67,000 new malware threats on the Internet every day. This means more than 45 new viruses, worms, spyware, and other threats were being created every minute – more than double the number from January 2009" (The Department of Commerce Internet Policy Task Force, 2011, p.ii).

Following the summary statistics of Hunt (2019) for the validation of compromised accounts in a data breach, recommended by the SOC team, seven of the top ten largest data breaches from cyber attacks are, at the time of the development of this work, registered between 2016 to 2019 with more than 3.2 billion accounts breaches, being the worst attack registered in January of 2019 with more than 772 million accounts breached from a list of 2.7 billion records. Which supports the severity and scalability of the damage volumes involved as time passes by.

Even though the numbers of reported damages are real, the true damage quantification remains uncertain or undisclosed as not only material costs in equipment and revenues are negatively affected. Three major factors can be identified:

- The organizational brand image perceived by the customer can be irreversibly damaged. Customer's trust and value are disrupted and comprised with unauthorized access of customer's personal information (name, personal identification, phone numbers, e-mail addresses, usernames and passwords, financial data) being extracted or publicly exposed (Bendovschi, 2015);
- As a consequence of the previous point, organizations are tempted to deny security exposures in order to minimize the damage and preserve a publicly positive sentiment (Center for Strategic and International Studies, 2014);
- The inability for most companies to discover that they have been compromised or data breached internally (Center for Strategic and International Studies, 2014). In fact, reports of security institutes and companies have estimated that only almost 30% of the organizations are able to do it by themselves and take an average of 205 days before detecting the presence of certain threats on their network (Wu, Lee, Wei, Hsieh, & Lai, 2017).

Among the business areas affected by this threat, one of the most sensitive and influential is the financial sector. Nowadays, this area is heavily dependent on the computer networks systems, relying on its technology for data communications and commercial purposes (Kumar et al., 2016). However, it is the financial sector that provides the best data on cybercrime because this sector is regulated, focused on cybersecurity and can easily measure their loss (Center for Strategic and International Studies, 2014).

Although the several obstacles to quantify the value of the damage, cybercrimes cause a substantial loss to the world economy. There are different approaches to calculate the costs caused by cybercrime. Comparing with the gross domestic product (GDP), the Center for Strategic and International Studies (2014) refers that cybercrime may be responsible for loss up to 1.5% of a country's GDP. The Center for Strategic and International Studies (2014) also reports different estimates for different types of countries: high-income countries can lose on average as much as 0.9% of GDP, while in developing economies the losses averaged 0.2% of GDP, being the average loss among all countries (that it was possible to obtain data) of 0.5% of GDP. The countries of G20, which produce the highest volume of wealth in the world, also suffer the greatest losses related to cybercrime and cyber espionage (Center for Strategic and International Studies, 2014). "The rate of loss from cybercrime was roughly the same (as a percentage of GDP) among three of the four largest economies in the world (the US, China, and Germany). These countries lost more than $200 billion to cybercrime" (Center for Strategic and International Studies, 2014, p.9). Thus, wealthier countries and business in North America, Europe, and Asia are more likely to suffer attacks, since they provide bigger returns than poor targets (Center for Strategic and International Studies, 2014).

In 2014, the Center for Strategic and International Studies (2014, p.6) used a different approach to estimate the annual global cost of digital crime and intellectual-property theft: "If we used the loss by high-income countries to extrapolate a global figure, this would give us a global total of $575 billion. Another approach would be to take the total amount for all countries where we could find open source data and use it to extrapolate global costs. This would give us a total global cost of around 375 billion dollars. A third approach would be to aggregate costs as a share of regional incomes to get a global total. This would give us an estimate of $445 billion". According to these statistics and knowing that internet economy generates profits between 2 trillion to 3 trillion dollars each year, cybercrime is responsible for loss between 15% to 20% of the wealth generated by the internet (Center for Strategic and International Studies, 2014).

Although none of these ways to calculate the global cost of cybercrime and cyber espionage is ideal, they are methods for estimating it. While reporting and data collection don't improve, the estimation costs will not improve either (Center for Strategic and International Studies, 2014). On the other hand, it is also difficult to calculate the real costs of cybercrime and cyber espionage, since there are intangible costs inevitable associated (Center for Strategic and International Studies, 2014).

Another problem is the difference between the value of what cybercriminals steal and the profit they can make with that (Center for Strategic and International Studies, 2014). "It is harder (in some cases, much harder) to monetize the result of a successful hack than it is to the hack itself" (Center for Strategic and International Studies, 2014, p.6).

In 2016, in financial services, the three more frequent patterns of cyber attacks were DDoS (the most common incident type), web application attacks, and payment card skimming (Verizon, 2017). The DDoS attacks are more frequent in organizations which use the internet to do business or communications (Verizon, 2017). The estimates point to a loss of revenue due to DDoS attacks over $10,000 per hour ($240,000 per day) among business organizations and over $100,000 per hour ($2,400,000 per day) for retailers (Neustar®, 2012). In the financial industry, 82% of companies lose more than $10,000 per hour during a DDoS attack (Neustar®, 2012; Verizon, 2017).

The increase of threats of internet forces the security policies, technologies and procedures to develop quickly and earlier, in order to prevent cyber attacks (The Department of Commerce Internet Policy Task Force, 2011). Cybercriminals are becoming more proficient and are constantly creating new ways to attack people, companies (in particular financial institutions) or countries (Rajan et al., 2017). "Protecting security of consumers, businesses and the internet infrastructure has never been more difficult" (The Department of Commerce Internet Policy Task Force, 2011, p.ii). In 2014, wide organizations such as Apple's iCloud, Op Albatross ATM thefts, Yahoo, PlayStation Network, and Microsoft Corporation suffered cyber attacks (Saad et al., 2016). Therefore, firms invest in different strategies to detect and prevent cyber attacks, through the acquisition of software and the creation of network security specialist teams to protect their networks (Jenab & Moslehpour, 2016). Therefore, it's estimated that in 2013 it was spent with cybersecurity products and services more than 58 billion dollars (Center for Strategic and International Studies, 2014).

2.4. CYBERCRIME AWARENESS

In recent years, several countries and organizations around the world have been increasing their awareness and concerns with cybercrime activities. In 2000, the Federal Bureau of Investigation (FBI) created a platform for the public in the United States known as the Internet Crime Complaint Center (IC3), responsible for receiving victim reporting crime complains as well as a public awareness channel for the population (Internet Crime Complaint Center, 2016). The annual Internet Crime Report published in 2016 by the agency, which provided some statistics from 2012 to 2016, registered over 1.4 million complains (on average 280.000 per year) and a total report loss of over 4.63 billion dollars, increasing year after year (Internet Crime Complaint Center, 2016). The report expresses concern with the real numbers and losses involved by referring that the reported figures are estimated to represent only 15% of the victims in the

United States (US) territory and an even smaller number when compared with the number of victims worldwide (Internet Crime Complaint Center, 2016).

In 2001, the Council of European created the first draft of the first international legislation against cybercrime, the Convention on Cybercrime (COC) (Hui, Kim, & Wang, 2017). The legislation sought international cooperation to fight cybercrime activities, promoting mutual assistance and providing legal framework to handle "any infringement against the confidentiality, integrity, and availability of computer data and systems, including common offenses such as distributed denial of service (DDoS) or malware attacks" (Hui et al., 2017).

In 2017, the World Economic Forum published its annual Global Risk Report (World Economic Forum, 2017). The report placed the technological dangers associated with large scale cyber attacks and massive incident of data fraud/theft, respectively in 6th and 5th places, in the list of risks most likely to occur in the next ten years. The reported technological risks predicted three important scenarios: i) a long-term pattern associated with the rising cyber dependency, expecting consequently an increase in the number of information infrastructure and network vulnerabilities; ii) widespread chaos, associated with large economic damages, geopolitical tensions or loss of internet trust due to large-scale cyber attacks or malware usage; and iii) unprecedented scale of incidents of data fraud/theft (World Economic Forum, 2017). Security concerns related to cyberwar and terrorism were also mentioned in the report, concerned with the usage of the cyberspace as a new domain of conflict used by nations and terrorist groups. The rising geopolitical tensions and violent extremist groups, associated with the rise of cyber attacks, major data breaches and hacks has led many countries to adopt new security measures and counterterrorism laws (World Economic Forum, 2017).

On the same year, Alhawamdeh (2017) published a research work proposing the development of a national institutional level information sharing framework to fight the cybercrime. The authors pointed out the existing cybersecurity information gap between countries, caused by the inexistence of a global framework platform for information exchange. Faced with information leakage dangers, each local authority works on their own in order to protect and manage their own information. The framework would provide the first layer of security protection tools used by countries, in a balanced commitment between security and data leakage.

3. LITERATURE REVIEW

3.1. BIG DATA ANALYTICS

Nowadays, Machine Learning and Big Data topics are subjects of great interest and focus among the scientific community (Nair, Shetty, & Shetty, 2017). A great variety of works developed have been continuously showing its potential and a wide range of applications.

Nair et al. (2017) have shown how predictive modelling of sensor data related to Oil and Gas Company can be performed through the Machine Learning platform H2O. In the same work developed by Nair et al. (2017), the authors discussed the usage of the online logistic regression for detection of phishing URL (Uniform Resource Locator), using the Hadoop framework and the scalable Machine Learning algorithms of Apache Mahout. The work also used Apache Storm to streaming data processing and WEKA classifiers for Machine Learning, as a phishing URL detection system. The social media data from social platforms, like Twitter, have been analyzed using Machine Learning in multiple research works with the objective of extracting useful information such as sentiments and tendencies of their users towards other persons or products, filtering of spam, finding trending topics, detecting real-time events like earthquakes, or personality prediction (Nair et al., 2017).

The constant growth on the number of internet usage has led to an exponential increase in network traffic (Kulariya, Saraf, Ranjan, & Gupta, 2016). New challenges have risen as the former tools and techniques are no longer efficient in processing the required volume of data (Gupta & Kulariya, 2016). New frameworks and software, like Hadoop and Spark (Kulariya et al., 2016), have been developed to handle the Big Data processing problem and provided the conditions for what is known as Big Data Analytics.

Big data Analytics is the combination of three different but interconnected scientific areas: i) Big Data Applications, ii) Data Mining and iii) Machine Learning (Epishkina & Zapechnikov, 2016). The first area, the **Big Data applications**, is a combination of a series of tools, techniques and approaches to effectively handle the current information explosion commonly described by the “three V” of Big Data: Volume, Velocity and Variety (Breier & Branišová, 2017). The concept behind the developed applications is heavily influenced by the parallel processing architecture of the Hadoop MapReduce model to process large datasets while handling parallelism challenges such as load balancing, network performance or fault tolerance (Epishkina & Zapechnikov, 2016). Most of the technologies around Big Data use or are influenced by software frameworks and libraries of the Hadoop and Spark projects under the development responsibility of Apache (Breier & Branišová, 2017; Epishkina & Zapechnikov, 2016).

The second area, **Data Mining**, is the name given to a series of methods and techniques used combined in a semi-automatic process of knowledge extraction from data (Breier & Branišová, 2017; Epishkina &

Zapechnikov, 2016; Fayyad, Piatetsky-Shapiro, & Smyth, 1996). Data Mining techniques result from a combination of various fields of study such as statistics, Machine Learning and database theory (Epishkina & Zapechnikov, 2016). Most of Data Mining methods have in its core the applications of Machine Learning algorithms and statistics methods to perform classification, clustering and regression tasks for knowledge extraction from data (Epishkina & Zapechnikov, 2016; Fayyad et al., 1996). Fayyad et al. (1996) define six of the most common Data Mining tasks, each described on Appendix 9.1: i) Association rule learning; ii) Clustering; iii) Classification and Regression; iv) Anomaly Detection; and v) Summarization.

According to Fayyad et al. (1996), based on web pools votings over the years, one of the main Data Mining process methodologies is the Cross-Industry Standard Process for Data Mining (CRISP-DM) (IBM, 2011). This process is comprised by six iterative steps, detailed on the methodology chapter: i) Business understanding, ii) Data Understanding, iii) Data Preparation, iv) Modelling, v) Evaluation and vi) Deployment (IBM, 2011).

The third area, **Machine Learning**, is a field of study focused on the development of learning algorithms capable of performing tasks of pattern recognition, data prediction or other decision-making tasks under uncertainty, without being explicitly programmed to do so (Epishkina & Zapechnikov, 2016; Murphy, 2012). As defined by Mitchell (1997), "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E". The algorithms are developed from feeding models with a dataset as input to make data-driven decisions as outputs (Epishkina & Zapechnikov, 2016).

Two main types of algorithms of Machine Learning are usually defined: i) predictive models, also known as supervised learning algorithms, and ii) descriptive models, also known as unsupervised learning algorithms (Murphy, 2012). The first ones, the predictive models, use as input a set of training examples previously labeled to perform predictions tasks over new and unlabeled examples, such as classification (if the target or labels are categorical or nominal values) or regression (if the target or labels are real-valued scalars) (Mitchell, 1997). The second ones, the descriptive models, have no targets to train using examples, instead, they use unlabeled data with the objective of identifying relevant patterns and structures among the data in tasks also known as knowledge-discovery (Mitchell, 1997; Murphy, 2012). A classic example of the usage of unsupervised learning algorithms is the clustering methods (Murphy, 2012).

The combination of the three areas provide Big Data analytics tools that allows large scale data collection, storage, processing and analysis using various techniques that fit to solve modern days problems in many areas from business, finance, healthcare, among many others, and with relevance in the information security and computer systems security area (Epishkina & Zapechnikov, 2016).

3.2. CYBERSECURITY ANALYTICS

The development of cybersecurity mechanisms to overcome increasingly complex threats issues has been a subject of increasing interest over the years, not only among organizations but also among the scientific community (Joseph, Laskov, Roli, Tygar, & Nelson, 2012). Over the years, an extensive variety of approaches, frameworks, techniques and algorithms have been developed and simulated for cybersecurity. The methods vary from information encryption, contents and services access control, and intrusion detection systems (IDS) among many others (Kulariya et al., 2016).

Some of the most interesting areas of application for Big Data analytics to cybersecurity problems are: i) Intrusion detection systems (IDS) and ii) Security information and event management (SIEM); among many other technologies (Epishkina & Zapechnikov, 2016).

The background for this work is the result of the combination of three interconnected areas of study related to security. The first area, where most of literature and investigation can be found, is the traditional IDS and the works developed using Data Mining techniques, with focus on supervised learning. The second area presents some of the approaches developed by IDS and Big Data analytics applications. The third and final area of interest is where the least amount of contributions is found, are the studies and works related to the development of intrusion detection on SIEM through supervised learning techniques to support system security. None of the studies in each of the areas completely fulfils the scope of this work but the three complete each other for this solution.

The rationale behind this organization of ideas is related to the objective of this work, the development of a detection model that is not intended to be used as a prevention method but a reactive one. The model developed in this work does not intend to substitute or prove its superiority in any way with the existing security systems, as each tool works as a piece of the security framework in an organization, and each piece is efficient on the purpose that it was assigned.

As security breaches are bound to happen in a real-world scenario (Quick, Hollowood, Miles, & Hampson, 2017), where client services and security restrictions are a delicate balance, the objective of this work is to fill an organizational need through the support of their security team, technically referred as Security Operation Center (SOC), on their investigations efforts, through one additional security layer, a threat detection system, over the data that has already been processed by the SIEM.

3.2.1. Intrusion Detection Systems (IDS)

The IDS are security software tools typically focused on detecting attacks or malicious traffic, classifying them as potential security threats or nonstandard behaviour events through the monitorization of the network, known as Network Intrusion Detection Systems (NIDS), and system activities or policy violations,

known as Host Intrusion Detection Systems (HIDS). In the case of the NIDS scenario, IDS can prevent the packet from being delivered or can alert or notify the SOC (Epishkina & Zapechnikov, 2016; Viegas et al., 2017). Traditionally, an IDS inspect the packet payload traffic in a network searching for potential security breaches or abnormal behaviour, usually supporting the antivirus, firewall (FW), access control and other systems in the security of the whole ecosystem. In the case of the HIDS scenario, the tools are capable of providing abnormal network usage recognition through the analysis of stored system logs (Fitriani, Mandala, & Murti, 2016).

According to Viegas et al. (2017), the typical IDS architecture results from the combination of four modules: i) Event gatherer – Responsible for reading and storing events from the system or network environment assigned; ii) Preprocessing – Responsible for all the work related with the parsing, transformation and feature extraction of the collected logs in order to ready them for the detection engine; iii) Detection - Module responsible for the previously processed event analysis for intrusion identification, classifying them as normal or as potential intrusion effort events; and iv) Alert – Module accountable for acting upon the events identified as potential intrusion, usually in the form of a generated alert or notification.

Several authors identify the detection methods of an IDS in two main distinct classes: i) anomaly-based, and ii) signature-based (Kulariya et al., 2016; Viegas et al., 2017). The first class, the **anomaly-based detection**, also known as behavior-based detection, is a method focused on the identification of abnormal behavior of the monitored traffic patterns typically through the comparison against previously analyzed traffic activities validated as normal (Fitriani et al., 2016; Kulariya et al., 2016; Viegas et al., 2017). This approach provides an effective advantage in the detection of new and unknown attacks through the deviation from normal traffic pattern behaviour (Fitriani et al., 2016). Fitriani et al. (2016) refer to the benefits of this approach in the prevention of DoS-based attacks, intrusion by a legitimate user or even Trojan horses. However, the downside of this method stands on the high number of misclassified anomaly events due to changes of the normal traffic pattern, resulting consequently in high false positive generated alarms (Fitriani et al., 2016).

The second class, the **signature-based** method, also known as **misuse-based** method or even **knowledge-based** method, is an approach of intrusion detection performed through the comparison of each of the events with a regularly updated database with information related with intrusion type patterns previously identified by a security expert (Fitriani et al., 2016; Kulariya et al., 2016; Viegas et al., 2017). The events with a matched signature with one in the signature database of threats are classified as an intrusion attempt (Viegas et al., 2017). This method is effective against known and stored intrusion types but is not able to generalize and identify a new threat, leaving the system exposed to new threats until the database is updated (Fitriani et al., 2016; Viegas et al., 2017). Additionally, the IDS can only inspect the traffic efficiently

if it is not encrypted. Most of the times the corporates cannot decrypt the traffic before analyzing with the IDS, therefore, what is analyzed is only the unencrypted traffic.

3.2.1.1. Data Mining on Intrusion Detection Systems

The IDS has been a research topic of great interest among the cybersecurity of network systems community (Kulariya et al., 2016). With the objective of suppressing the traditional disadvantages of the IDS, several works have proposed solutions that combine the previously functionalities and advantages of the detection systems with Data Mining and Machine Learning approaches. Epishkina and Zapechnikov (2016) suggests, as an example, the usage of Data Mining methods for intrusion detection in several potential ways, in which the following stand out: i) Classification; iii) Clustering; and iii) Anomaly Detection.

The first method,, the **classification**, is described by Epishkina and Zapechnikov (2016) as an evident intrusion detection method. Through the collection of known intrusion attempts, it is possible to train models to be able to classify new threats from new unclassified instances. The author suggests the potential usage of a decision tree, neural networks, Bayesian classifiers, support vector machines (SVM) and genetic algorithms. Breier and Branišová (2017) use, as the basis for its work, comparative studies of classification tasks over log files using decision trees, neural network, Naive Bayesian and Support Vector Machine.

The second method, **clustering**, is described by Epishkina and Zapechnikov (2016) as a useful approach with potential applications, for example, on the creation of intrusion signatures clusters merged with alerting functions in order to produce an alerting system capable of identifying potential attacks whose behaviour is similar with the created clusters.

The third method, **anomaly detection**, is one of its most interesting applications for intrusion detection as an unsupervised learning approach, using clustering and density algorithms to define patterns of events with similar behaviours and assumed to fit normal behaviour. New instances unfitting or too sparse from any of the previously created clusters will be labelled as abnormal events and potential threats. Epishkina and Zapechnikov (2016) suggest the application of these methods in network traffic data packet headers, such as Ethernet, Internet Protocol (IP), Transmission Control Protocol (TCP), as features for the definition of attack-free clusters. Posterior new traffic packets not similar to any of the clusters would be considered anomalous.

Several authors explore in their works the potentials and advantages in the usage of Data Mining and Machine Learning methods applied to IDS. Kulariya et al. (2016) argue that the variability and speed in which a lot of new attacks are generated every day, defines two fundamental characteristics of any IDS: adaptivity and fast detection capabilities, in order to detect new attacks. The usage of Data Mining and Machine Learning methods are able to provide these critical characteristics to the IDS (Kulariya et al., 2016).

Through pattern extraction of normal and malicious data records, it is possible to create and train different classifiers capable of identifying different types of attacks (Kulariya et al., 2016).

In 2012, Nadiammai and Hemalatha (2012) referred, the already mentioned, potential threats that came along with the increasing usage and evolution of the internet. The work had the objective of applying Data Mining algorithms to intrusion detection. The author described the comparative study of several rules and function-based classifiers performances, namely Part, Ridor, NNge, DTNB, JRip, Conjunctive Rule, One R, Zero R, Decision Table, RBF, Multi-Layer Perception and SMO algorithms, through the metrics of accuracy, specificity and sensitivity over the KDDCUP'99 dataset. The work concluded that the sequential minimal optimization algorithm (SMO) and the NNge algorithms where the most promising ones regarding the dataset used and performance metrics indicated.

In 2013, Chauhan, Kumar, Pundir, and Pilli referred the importance of the potential role that Data Mining approaches could provide on the development of IDS over network traffic and presented a comparative study on the ten most promising classification algorithms, selected out of the twenty most widely used classification algorithms. The work assessed the accuracy, specificity and sensitivity, and training time of J48, BayesNet, Logistic, SGD, IBK, JRip, PART, Random Forests (RF), Random Tree and REPTree algorithms, using the NSL-KDD dataset. The study concluded that the Random Forests algorithm had the best performance with respect to accuracy, specificity and sensitivity, while the IBK algorithm took the least time to train. The author left as a future work the possibility of combining different Data Mining algorithms and data reduction techniques to reduce the rate of false negatives (FN) and increase the overall accuracy (Chauhan et al., 2013).

On the same year, Nagle & Chaturvedi (2013) presented a work exploring the combination of an IDS with the implementation of a classifier algorithm for the security network detection activities. The work compared the implementation several classifier algorithms, Naïve Bayes, Bagging, Boosting, Stacking, and J48, on different attack types, using the NSL-KDD dataset and the feature reduction technique of the information gain (IG). The authors concluded that, depending on the attack types, the J48 classifier presented the best performance in the intrusion detection tasks, while the Stacking classifier presented the worst.

In similar but improved approach, Prachi (2016) compared a wide variety of classification techniques in order to identify a Machine Learning algorithm capable of providing both high accuracy and real-time system application for intrusion detection on network traffic. The author explored the optimization of the IDS to face against the increasing volumes of network data and the complex nature of intrusions. As such, the work sought for maximum accuracy and minimum model building time in order to be able to perform in real-time IDS. The work evaluated fifteen different classification algorithms, such as Naïve Bayes, Logistic

Regression, Jrip, J48, Random Forest, Random Tree, among many others, and the KDDCUP'99 and NSL-KDD dataset. The author concluded for the algorithms tested and the indicated datasets, that the Random Forests algorithm had the highest detection rate and lowest false alarms in comparison to other algorithms. However, it also took significant time to train, making the Random Tree the chosen algorithm for its significant high detection rate and minimum model building time, to be implemented as a real-time IDS. The author focused on the training time of the algorithms and left as a future work the real-time implementations and performance assessment.

However, early detection is not always guaranteed against security threats as security breaches keep occurring in a never-ending spiral of complexity and variety of approaches either from the perpetrators and security countermeasures.

3.2.1.2. Big Data Analytics on Intrusion Detection Systems

As referred before, the exponential increase on the internet usage in the present days as lead network traffic data sizes and variety to a point where tradition data processing engines can no longer handle it efficiently (Gupta & Kulariya, 2016). These challenges are particularly critical in sensitive areas such as the cyber-security. Tradition IDS or even anti-virus can be exploited by perpetrators with ease due to the plain amount of network traffic exchanged every day (Mahmood & Afzal, 2014).

Mahmood and Afzal (2014) identify some of the main problems related to cybersecurity and the Big Data challenges:

- Organizations growth in products and services using the internet as a commercial platform, through computers, mobiles or even clouds, lead to the natural increase in data and information exchanged between clients and organizations and between organizations. This ease of data accesses also led to more and varied network vulnerabilities and thus contributing to the cyber attacks.
- The increase in data volume, variety and complexity as provided ground for the increase in hacking skills with new approaches and new opportunities, making traditional security systems inefficient (e.g. the traditional signature-based tools).
- The exponential growth in the volume of today's traffic network data lead to two scenarios, either the excessive amount of security information alerts to be handled by security experts or only a small slice of the security information is collected for analysis.
- Traditional computer hardware and software architectures are not efficient to process and analyze the variety, complexity and speed in which Big Data transacts from different sources, different storages and different machines.

As referred previously, several authors argue that cybersecurity detection systems should not only be accurate and adaptive but fast and efficient in dealing with the variable nature, complexity and size of the network traffic (Gupta & Kulariya, 2016; Kulariya et al., 2016).

In 2015, the online publication of Breier and Branišová (2017) presented a work exploring the possibility of exploiting network activity log files from various network devices to identify security breaches. The authors proposed the implementation of Data Mining techniques for dynamic rule creation in an IDS, supported by the parallel storage and processing of the Apache Hadoop framework in order to handle the huge amount of data containing within the files. The final result demonstrated that the model was capable of detecting new types of intrusions with an acceptable error rate while keeping competitive speeds when compared with the FP-growth and apriori algorithms.

In the year of 2016, Gupta and Kulariyas proposed a framework for fast and efficient cybersecurity network intrusion detection using Apache Spark, an open source cluster computing platform designed for parallel processing, and its MLlib library to perform classification and performance assessment tasks over the logged network traffic data. The work compared two feature selection methodologies, correlation-based feature selection and hypothesis-based feature selection as well as five Machine Learning algorithms for the classification problem: i) Logistic regression, ii) SVM, iii) Random forest, and iv) Gradient Boosted Decision trees.

In the same year, the size and complexity of network traffic lead Kulariya et al. (2016) to explore a Data Mining solution capable of supporting real-time intrusion detection. The author argued that the Machine Learning algorithms implemented should not only be efficient and accurate on detection of attack traffic but also fast and scalable. The work used two correlation-based feature selection and chi-squared feature selection and compared the performance of five classification algorithms namely the Logistic regression, SVM, Random forest, Gradient Boosted Decision trees and Naive Bayes. The comparison used the Apache Spark framework for the parallel processing and the real-time network traffic dataset of KDDCUP'99. The algorithms were evaluated not only in the metrics of accuracy, sensitivity and specificity but also in its training and prediction time. The authors concluded that the Random Forests algorithm provided the best accuracy, sensitivity and specificity results, while the Naive Bayes algorithm provided the worst specificity performance but required the least time to train.

3.2.2. Security Information and Event Management System (SIEM)

In 2016, Suh-Lee, Jo, and Kim referred the importance of the exponentially growing messages generated by the computer systems and applications in a modern computing environment and their potential as a source of information useful for advanced threat detection (Suh-Lee et al., 2016). The authors give

emphasizes to the volume, variety, and complexity of data generated in logs as a hard and complex task for security analysts. Breier and Branišová (2017) referred to the size of log data generated every day is expected to grow more and more as time passes by and technology continues to evolve.

The SIEM are software solutions developed as a natural response for the great amounts of log data generated every day throughout the complexity of interconnected and distributed computer systems that support organizations (Suh-Lee et al., 2016).

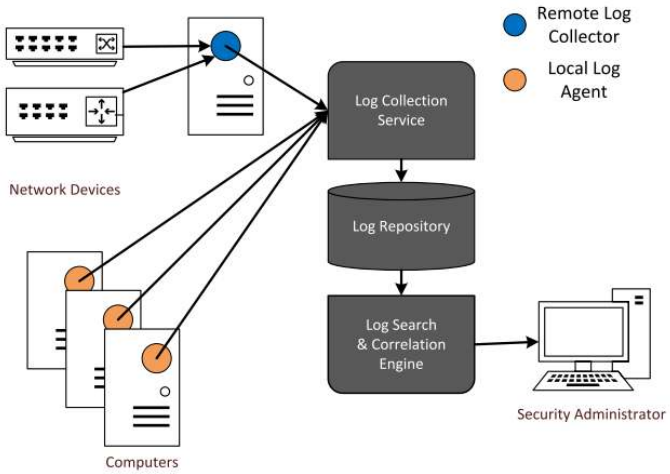


Figure 3.1 - SIEM Architecture (Suh-Lee et al., 2016)

The SIEM are solutions designed to (Epishkina & Zapechnikov, 2016; Lee, Kim, Kim, & Kim, 2017; Suh-Lee et al., 2016): i) Centralize and manage system-generated log messages, also known as Syslog data, in one central location through log collection and aggregation from multiple data sources; ii) Preprocessing (parsing and normalizing log data); iii) Log storage and retention of historical data for a specified period of time (particularly useful for organizations due to legal requirements to which they are subjected in the retention of electronic records of transactions); iv) Search log data; v) Alerting and dashboarding services; and vi) event correlation of log messages.

These tools are capable of providing security support through the correlation of the generated logs such as the FW, IDS, anti-malware systems, authentication services, HTTP servers, corporate computers, and others that are deployed at both on the host and on network domains (Epishkina & Zapechnikov, 2016). According to Suh-Lee et al. (2016) the correlation engine is capable of supporting the security systems through application of predefined correlation rules on logs for threat and anomaly detection, taking advantage of the centrality and normality of the logs collected, in order to gain better situational awareness of the intruder's attempts in the systems (Suh-Lee et al., 2016).

The work presented by Suh-Lee et al. (2016) identifies some of the SIEM's technology and usage advantages and disadvantages. The author highlights the importance of this technology as the current state-of-art in its main functions of log management, aggregation and storage. It is also capable of providing important

support to the security systems through the effective identification of some potential threat events, through its rule-based detection scheme, and event correlation engine, using predefined correlation rules over the preprocessed and structured logs it outputs. However, Suh-Lee et al. (2016) identify three important technology limitations or unexplored potential in its usage as a security system support tool: i) Underutilization of the event detection engine; ii) Loss of opportunity in the exploitation of unstructured messages generated by the system; and iii) The rule-based detection is limited and dependency of expertise validation.

The first one, the underutilization of the event detection engine, is described by Suh-Lee et al. (2016) as a problem related with the high cost, time-consuming and in human resource expertise, for implementation of new, and more accurate and specific security operations for the detection. The expertise required to handle the complexity associated with manually creating rules, the rigidity of the implemented rule-based detection algorithms, and the deterministic parsing schemes require that only logs that follow certain logging protocol are suitable to be processed, create a discouraging obstacle for organizations and researchers explore further the detection functions within these tools (Suh-Lee et al., 2016).

The second limitation, the loss of opportunity in the exploitation of unstructured messages generated by the system, is the main focus of Suh-Lee et al. (2016) work. The authors argue that the system is only capable of retrieving insights from structured logs preprocessed by its parsing engine, wasting the potential information within the unstructured part of the log message. The current deterministic log parsing scheme is not suited for processing and working on "free-text pseudo-natural language" messages produced by the system (Suh-Lee et al., 2016, p. 253). Providing that the detections and correlation engines act upon parsed structured logs, the unstructured logs will be unused causing a significant limitation on its depth of information used for correlation of different events. Most logs that are fed to SIEM are not structured to be processed by SIEM, unless they are parsed previously, they will be collected as unstructured data, resulting in inefficient usage of the correlation engine of the data logs.

A third limitation related to the usage of SIEM as security system support tool can be argued related with the rule-based detection system itself. As previously seen in the IDS, this threat detection approach is effective in the detection of previously analyzed information by security experts. A rule database is continuously updated upon the discovery of new threats by security experts, either within the organization or from an external organization. This inability to adapt to new threats and dependency of expertise validation all the time can be a serious limitation in some organizations and services. The lag between threat event occurrence and the response time (discovery of a new threat, the validation and rule creation by a security expert and the update of the rule database) is a threat in a way that creates an exposure window that some critical organizations cannot afford.

3.2.2.1. Data Mining on SIEM

According to Epishkina and Zapechnikov (2016), the SIEM systems functionalities are a potential and promising field of research, as most of its functions can be significantly improved through statistical and Data Mining models.

Most of the security research developed explores the potentials of the Data Mining models focus on the IDS systems. The few projects related with the SIEM systems and security usually aim to provide primarily solutions to overcome the complex challenges of processing and retrieving important information from the unstructured logs, through unstructured log analysis and text mining, and only then the implementation of Data Mining and Machine Learning models over the most relevant extracted features.

In 2013, Azodi, Jaeger, Cheng, and Meinel presented work focused on the development of a system that combines the capabilities of IDS and SIEM systems (Azodi et al., 2013). The work focused initially on handling some of the challenges of the heavy processing associated with the unstructured nature of the log data. The author states that the processing and analysis of the event information that the systems receive can seriously deprecate the speed and accuracy of event correlation engines, which is vital for real-time analysis. The work presented a method to improve the system performance through the detection of the input log type and format using regular expressions and normalizing log entries. The extracted information from the unstructured logs was then used to develop a rule-based detection engine to perform security tasks.

This work was followed by the publication in 2016 of Suh-Lee et al. (2016), which proposed a different approach for the usage of unstructured logs in SIEM for threat detection. The author used text mining and natural language processing, instead of regular expressions, to handle and exploit the unstructured log potential information regarding threat detection. The extracted information was then fed to Machine Learning models to train and perform threat detection tasks instead of a rule-based engine like in the previous work. The work explored the performance of twelve classification algorithms such as the J48, Naive Bayes, Naive Bayes Multinomial, Voted Perceptron, AD Tree, Random Forests and Random Trees, using different extracted information formats. To achieve the task, the authors used the simulated attack data from the SAIKON 2006 IARPA dataset as well as two different attribution selection algorithms, the Information Gain (IG) and the chi-squared test, and the 10-fold cross-validation method. The performance was evaluated through the metrics of accuracy, precision, recall, specificity and training time. Among the work's conclusion, some of the most interesting are related to the performance of classification algorithms. For the SAIKON dataset using both messages and extracted features, the tree-based algorithms of Random Forests, Random trees algorithms and J48 produced the best results.

Both works developed important solutions for what is recognized as a complex and quite unexplored task of using SIEM systems for threat detection, using the information extracted from their unstructured log messages. The work developed by Suh-Lee et al. (2016) presented a particularly interesting approach with the implementation of Machine Learning algorithms to assess intrusion events registered in the SIEM log messages.

One important difference between the present work previously presented by Suh-Lee et al. (2016) is the format in which the log data was initially received. While both works dealt with the hardships and unexplored hidden potential information of the unstructured log messages of SIEM, the data used in this work is from a practical real-life scenario and benefits from the already parsed logs by the SOC team, resulting in a dataset of structured logs messages from the start. Aside from the way the information is handled to transform unstructured logs into structured ones, the following general steps related to the implementation of classification algorithms for threat detection are similar. The work of Suh-Lee et al. (2016) explored the usage of the classification algorithms available on the Data Mining software WEKA from a local processing standpoint, while this work explored a different technological stack from a Big Data distributed processing point of view with their respective available learning algorithms. Most of the log features extracted to train the Machine Learning algorithms for intrusion detection in those works were extracted for this work.

4. METHODOLOGY

The methodology chapter comprises the definition, description and justifications behind all the methods, approaches and techniques used for the development of this work. Firstly, the distributed system for parallel processing environment setup used for this works is described, along with all its most relevant technologies involved. Next, the overall Data Mining methodology used is described and detailed. For each step and for every decision made and method implemented over the data of this work, the results achieved are presented along the way. The final results tough, are presented and discussed in the following chapter.

4.1. DISTRIBUTED SYSTEMS / PARALLEL PROCESSING

The steep growth in data collection, storage, processing and analyzed over the years lead the technological development to a world of new opportunities but also new challenges in what is called the Big Data era (Fu, Sun, & Wang, 2016). Over time, the volume, variety, velocity and veracity (Peña, 2017) data properties present in Big Data environments reached a threshold where the present technological systems had to evolve either by scaling up to high-performance computing with supercomputers or by scaling out to distributed systems built on multiple commodity machines (He, Zhu, He, & Lyu, 2016). Being the later one a promising solution regarding the trade-off between computational power, scalability and economic investment, several technologies have been developed focused on handling massive data volumes by distributing data storage and processing over a cluster of machines.

4.1.1. Apache Hadoop Framework

Several open source technologies capable of providing distributed computation and storage across a cluster of machines are available, where Apache Hadoop stands as the most widespread (Fu et al., 2016; Vavilapalli et al., 2013; Wisesa, Ma'sum, Mursanto, & Febrian, 2016).

The Apache Hadoop's development is managed by the Apache Software Foundation and provides distributed computation over a cluster of commodity hardware computers of large data volumes, by allocating computation and storage tasks to each machine (Apache Software Foundation, 2018e; Wisesa et al., 2016). On top of the distributed processing, the reality and complexity of the different agents involved in the tasks lead to the assumption that hardware failure is to be expected at some point (Apache Software Foundation, 2018e). Therefore, the framework provides mechanisms for delivering high-availability throughout the cluster machines in a reliable and fault-tolerant manner, actively assessing each node and handling the failures at the application layer (Apache Software Foundation, 2018e).

Among the basic modules available that support the distributed processing tasks on Hadoop framework, such as the Hadoop Common and Hadoop Ozone, the following stand out as the core of the framework

relevant for this work (Apache Software Foundation, 2018a): i) Hadoop Distributed Files System (HDFS); ii) Hadoop Yet Another Resource Negotiator (YARN); and iii) Hadoop MapReduce.

The first module, the **HDFS**, is a distributed file system inspired by the Google File System (Ghemawat, Gobioff, & Leung, 2003) and was designed to offer high throughput access to application data over the cluster machines, while providing a reliable and highly fault tolerance storage of the data, using an architecture comprised of a NameNode (master node) and multiple DataNodes (slave nodes) (Apache Software Foundation, 2018e). An in-depth description of HDFS technology and architecture is provided on the Appendix 9.2.

The second module, the **Hadoop YARN**, is a framework designed for cluster resource management and job scheduling (Apache Software Foundation, 2018b; Vavilapalli et al., 2013). The processes are performed by two entities in what is referred to as the "data-computation framework" (Apache Software Foundation, 2018b): i) the ResourceManager (RM) for global management of the cluster resources among all applications submitted to the system using the Scheduler, responsible job scheduling activities based on the resource requirements of the applications and resource allocation itself to the various running applications using the abstract notion of resource container, and ApplicationManager, tasked to accept application submissions, perform the negotiation procedures for launching the Application Master (AM) on the first container; ii) NodeManager, an agent present in each DataNode, responsible for controlling the resource containers, their resource consumption monitoring (CPU, memory, disk, network, etc) and respective reporting to the RM/Scheduler (Apache Software Foundation, 2018b; Vavilapalli et al., 2013). An in-depth description of the YARN technology and architecture, and workflow is provided on the Appendix 9.3.

The third module, the **Hadoop MapReduce**, is a programming paradigm implemented as a framework for easily writing applications capable of performing parallel processing large volumes of data over the machines of the clusters (Apache Software Foundation, 2018f). The MapReduce application implementation is typically divided into two stages (Apache Software Foundation, 2018f; Dean & Ghemawat, 2008): i) Map tasks, comprised by the data splitting and mapping; and ii) Reduce tasks, comprised by the data shuffling, sorting and reducing. A more detailed description of the MapReduce process is provided on the Appendix 9.4.

The MapReduce implementation is usually integrated with a RM, in this case, the YARN. Upon receiving a Hadoop job submitted by the client and the respective configuration, the RM is responsible for distributing the software/configuration to the worker nodes, support and orchestrate the scheduling and monitoring tasks, as well as their re-executions in the case of failure of all the jobs submitted by the client (Apache Software Foundation, 2018f). Thus, typically the architecture is comprised by a single master node with the

RM, one NodeManager for each slave cluster-node, and for each launched application an ApplicationManager (Apache Software Foundation, 2018f).

Several other Hadoop-related projects have been developed and implemented on top of the main basic Hadoop framework. Some provide new and different capabilities, others are technological enhanced and better versions of the previous ones developed to overcome bottlenecks and shortcomings. Among the projects, one stands as the most relevant for this work, the Apache Spark Framework.

4.1.2. Apache Spark Framework

The MapReduce processing engine was the pioneer model for parallel computation over a cluster of commodity hardware while providing automatic task scheduling, fault tolerance and load balance. However, another framework was later developed with the objective of solving some of its shortcomings while retaining most of its benefits, something that to the date of its development had not been achieved, the Apache Spark Framework (Zaharia et al., 2012).

The development of the Apache Spark Framework is managed, like the Hadoop Framework, by the Apache Software Foundation. According to Zaharia et al. (2012) reports from Hadoop users have identified deficiencies related to applications that are inefficiently implemented using the acyclic data flows of the MapReduce. Applications that rely on iterative computing jobs fall short in performance if for every iteration a job is generated, and the data has to be reloaded from the local storage, such as in many Machine Learning algorithms where during an optimization problem a set of functions are iteratively applied to the same dataset in order to optimize a set of parameters (Wisesa et al., 2016; Zaharia et al., 2012). The authors Fu et al. (2016) reinforce the problem and identify the high overheads at the launch of each job and the dependency of physical storage to support the processing jobs as the main time constraint bottlenecks.

The Apache Spark Framework is an open source cluster computing framework that provides a distributed data processing engine optimized for low-latency tasks while using the memory to store the intermediate and output results of the processed jobs (Fu et al., 2016; Gupta & Kulariya, 2016; Joglekar & Pise, 2016; Zaharia et al., 2012). According to Zaharia et al. (2012), Fu et al., (2016) and Gupta and Kulariya (2016), by using a memory computing solution, Spark uses the benefits of parallel processing from the Hadoop MapReduce approach and improves the efficiency of the data computing processes and computation variations, promoting a more performant implementation of iterative processing natured applications such as the ones used for Data Mining and Machine Learning.

The Spark in its core presents a processing engine that relies on an abstraction called resilient distributed datasets (RDDs) (Zaharia et al., 2012). According to Zaharia et al. (2012) an RDD is defined as a “read-only

collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost". These objects can be cached in memory in a distributed manner over the machines of a cluster in order to be reused multiple times in parallel processing operations, thus promoting the iterative natured applications implementations (Zaharia et al., 2012). The RDDs objects provide fault tolerance during the parallelized processes through lineage. Before the failure scenario of an RDD, the information in it is no longer accessible, however using the concept of lineage, where the transformations used to create the RDD from other datasets is memorized, an RDD can be rebuilt.

Spark can be run integrated with the Hadoop cluster ecosystem tools, such as the access to distributed storage systems like HDFS or use the YARN as a RM (Gupta & Kulariya, 2016). The framework is an ongoing development project that supports, among other implementations, applications with batch and iterative processing applications, iterative queries as well as streaming while providing an API for different programming languages, such as Scala, Java, Python and SQL (Gupta & Kulariya, 2016; Wisesa et al., 2016; Zaharia et al., 2012).

Among the ecosystem of projects that comprise the Spark framework, the following modules stand as the most relevant for this work: i) Spark SQL module; and ii) Spark ML/MLlib packages.

The **Spark SQL** module introduces the concepts of structured data processing with a new abstraction, the DataFrames (Apache Spark, n.d.-e; Armbrust et al., 2015). A Dataframe is a distributed collection of records organized in named columns, similar to a table from a relational database or the pandas dataframes used in Python (Apache Spark, n.d.-e; Armbrust et al., 2015).

According to Armbrust et al. (2015) "Spark SQL provides a DataFrame API that can perform relational operations on both external data sources and Spark's built-in distributed collections". Therefore, DataFrames can be created from structured data files, such as CSV files and existing RDDs, or from running SQL using another programming language, such as Python, to query tables in Hive or other external databases (Apache Spark, n.d.-e; Armbrust et al., 2015). The DataFrame structures are also integrated with other Spark projects, such as the Machine Learning package Spark ML that take them as input and output data formats (Apache Spark, n.d.-e; Armbrust et al., 2015). Additionally, using the information related with the structure of the data and computation being performed over a DataFrame, Spark SQL operations are supported by Catalyst, a relational optimizer that enhances Spark's performance over relational operations, such as querying (Apache Spark, n.d.-e; Armbrust et al., 2015).

According to Apache Spark (n.d.-b), the Spark Framework presents two libraries for Machine Learning purposes, the **Spark ML** (API built on top of DataFrames) and the **Spark MLlib** (API built on top of RDDs), that provide access to Machine Learning algorithm implementations either for supervised (classification and

regression) and unsupervised learning (clustering), model assessment, selection and tuning, feature extraction, transformation and selection, ML pipelines for model building, among many other functionalities to be used in a distributed processing manner.

4.1.3. Experimental Setup

In this subchapter, the experimental setup for the development of a threat detection distributed processing framework for the prediction of threat server requests is defined. For all the experiments, tests, developments and models of this work, the used hardware and software characteristics are indicated along with its technical specifications as follows:

- Hadoop cluster with four nodes, one Namenode (master) and three Datanodes (worker nodes). Four machines were used with 32GB RAM, 2TB Disk HDD, a processor AMD Ryzen 5 1500X Quad-Core with 8 logical cores, and each machine used one VM with a CentOS Linux distribution, 16GB of RAM, 6 processing cores, and a 1TB Disk;
- The full services stack installed and used for this work comprised the services for Ambari, HDFS, YARN, Hive, ZooKeeper and Spark, all described on the Appendix 9.5;
- For programming language Python 2.7 was used to interact with the Spark API through PySpark;
- For all the processes in this work related with data ingestion, data preparation, Machine Learning modelling and evaluation processes, Spark SQL module (Apache Spark, n.d.-d) and Spark ML package (Apache Spark, n.d.-c) was used with Spark DataFrames as the main abstraction;
- For cluster configuration and Spark-Submission parameters, the combination displayed on Table 4.1 were used, combining the number of nodes with tiny, fat and balanced executors Grover & Malaska (2016).

Table 4.1 - Experimental Cluster Architectures and Spark Parameter Configurations

Spark-Submit Parameters	1 Worker Node	2 Worker Nodes			3 Worker Nodes		
		Tiny Executors	Fat Executors	Balanced Executors	Tiny Executors	Fat Executors	Balanced Executors
master	local [6]	YARN	YARN	YARN	YARN	YARN	YARN
deploy-mode	-	client	client	client	client	client	client
num-executors	-	12	2	3	18	3	4
executor-cores	-	1	6	3	1	6	3
executor-memory	-	3GB	16GB	6GB	3GB	16GB	6GB
drive-memory	3GB	3GB	3GB	3GB	3GB	3GB	3GB

4.2. DATA MINING METHODOLOGY

The development of this work followed the Data Mining methodology proposed in the CRISP-DM (IBM, 2011). According to Fayyad et al. (1996), based on web pools votings over the years, one of the main Data Mining process methodologies is the CRISP-DM (IBM, 2011).

The process is comprised of six iterative steps: i) **Business understanding** (detailed definition, identification and understanding of the business problem); ii) **Data Understanding** (data collection and exploration); iii) **Data preparation** (all the data manipulation steps required to transform and create a final dataset to feed the models of the next stage); iv) **Modelling** (applying Data Mining and Machine Learning methods and optimizing parameters to fit the best model); v) **Evaluation** (Assessment of the models performances with appropriate metrics with respect to the business goals); and vi) **Deployment** (full implementation of the data collection, preparation and modelling framework with the best model) (IBM, 2011).
From the previous, the deployment will be out of the scope of this work as it would represent an actual fully integrated implementation of the solution into the organization's services architecture, with consequences at business and operation level. The scope of this work is to prove the added value of the solution.

4.3. BUSINESS UNDERSTANDING

With long experience in the financial area, the financial organization of this work operates in the payment systems industry with the mindset of being an international reference. For confidentiality reasons, the financial service, organization identification or any sensitive information were obfuscated or removed from this work.

The mission of the company is to contribute to the wellbeing of the society, promote efficiency in its financial services, mainly in the payment areas, by proving technological solutions and processes that combine security, convenience and innovation at the least cost possible. The financial organization provides services between other financial organizations and clients. The clients can be from merchants to end users. For many years, the financial organization was mainly focused on processing a financial transaction. Recently, it has been providing new services regarding new technologies to add value and create new opportunities, like services using mobile applications. The sensitive nature of the information handled by the company justifies the importance and the need of having all the infrastructure, services and processes well defined and protected. Efficiency and constant adaptation to new technologies, regulations and attack vectors, is the key for a financial company to be competitive and successful. As the technologies grow and new services are provided to the clients, new techniques and processes need to be developed in order to have a security team capable of monitoring and perform analysis over all the potential attacks. Due to the sensitive nature of the information handled, the organization is certified in the highest ranks of security over the several layers of security and safeguarding its safety against intrusion attempts.

The focus of this work is a specific financial service used for international payments, either by mobile or at a point of sale (POS) through merchants. All the service requests to the organization's services are collected by

servers, monitored through several layers of security and followed closely by SOC, a team composed by security experts.

The complexity and high volumes of information flow generated by the logs of every instance of the organization's environment are centralized by SIEM, which is responsible, as seen before, for the collection and correlation of the system's applications and services log data providing a hawk eye over the system. At the SIEM level, the information regarding server requests over any service is not as detailed as the one extracted from the service directly. However, the centralized view of the system provides an enriched and integrated view of financial service server requests data flows. Moreover, the information is accessed unencrypted providing a different perspective of the server request when compared with other system configurations of the Organization such as the information controlled at the Intrusion Detections Systems (IDSs) security layers. Therefore, regardless of the intrusive server requests being efficiently blocked right at the IDSs or FW level, the information extracted from the SIEM provides the ground to open deeper investigations regarding the root cause analysis of the intrusions, integrating every system layer.

Currently, and in accordance with the implemented set of manually defined ruled-based system, the SOC can only open investigations focused on intrusive server requests attempts with high expression over the volume of information transacted. Due to the limited available human resources of experts, it is impractical to open investigations overall intrusion attempt, ending in an impossible tracking scenario of the number of false positives or negatives involved, as the process of opening investigations is not managed efficiently. The steep growth in the volume of information from the increasing use of different financial services demands a more versatile, efficient and automatic solution as the manually defined rule-based system ends up being efficient but not updated fast enough for the complex and changing nature of the intrusion attempts.

4.4. DATA UNDERSTANDING

The data understanding stage is comprised initially by the data collection process, followed by the data exploration through statistical analysis and data quality validation. This phase has the purpose of framing the data contents to our business problem, acquire insights related to the available data and identify the required preprocessing steps to take on the next stage.

4.4.1. Data Collection

The data collection definition is the first step towards understanding the content of the data. Understanding the data source and all the data flow associated with the business process is a critical component in linking the business problem to the data problem in hands to be solved.

The diagram displayed in Figure 4.1 represents the data flow from source requests to the security operation team.

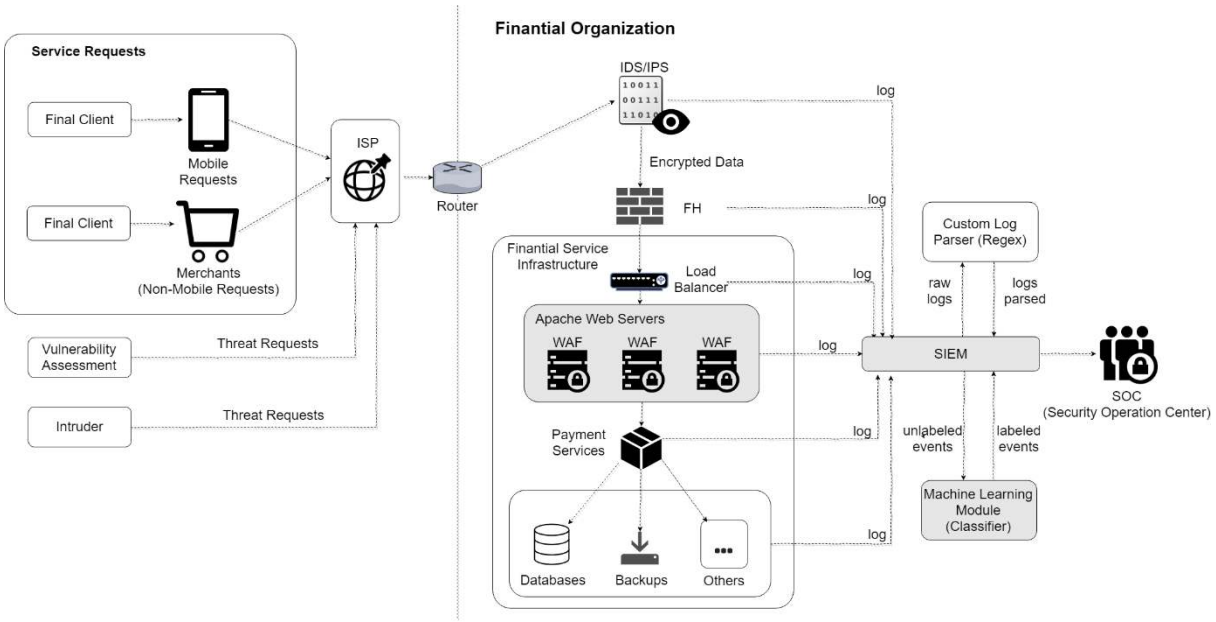


Figure 4.1 - Financial Organization Data Flow

The organization has a financial service on the international market that allows clients through mobile devices or through merchant POS devices to make service requests. A third and fourth source of the request are received, respectively, from a certified company responsible for periodical assessments of the organization's services system vulnerabilities with the latest rising security exploits, and actual abusive intruder requests with intentions of exploiting system vulnerabilities and cause harm. All the requests are received by an Internet Service Provider (ISP) and redirected to the financial organization. The entry point is a router that receives the encrypted information from the request and redirects it to the infrastructure responsible for handling the service request. Throughout this process, the requests are inspected by an IDS and filtered by the FW. The load balancer receives the requests unfiltered to this point and redirects them to Apache Web Server with most resources available. For each Apache Server layer, the requests are validated by a Web Application Firewall (WAF). All the structures that serve the payment service, from databases to backups systems and others will feed on the content of the Apache web servers. For each of the previous steps, logs are produced with reports related to the content received, analyzed and outputted. The logs produced are collected in a raw format through the SIEM system. Each raw log event is then parsed by the security team using custom regex-based processes to extract all the fields that were considered valuable for the company. In the endpoint is the SOC team that receives the parsed logs from every system infrastructure to perform monitoring, data correlation and human investigations over triggered potential threat requests.

The dataset provided contains a column created for the purpose of this work, the target variable. It is a binary variable that contains the classification of each historic log server requests, as a potential threat (labelled 1) or a normal service request (labelled 0). The target variable was created through the

combination of two data sources: i) Manually identified server request threats by the security team; ii) Penetration tests from an external and certified company responsible purposely search for vulnerabilities and assess the organization's security against a continuously updated list of worldwide identified security vulnerabilities. The first source is the result of the continuous efforts of the SOC team on their continuous work of monitoring and investigating potential threat events. The second source, the penetration tests, represent the majority of the events identified as threats as they are performed periodically every week in a batch of independent server requests of the service. The tests are executed from a fixed range of IP addresses, allowing their mapping and classification as threat requests to the server.

The continuous and steep growth of the log server requests over time, since the financial service was launched internationally, is a matter of concern by the organization. As displayed in Figure 4.2, the dataset provided has grown over time from 1.4 million to more than 10.2 million server requests by month in less than a year.

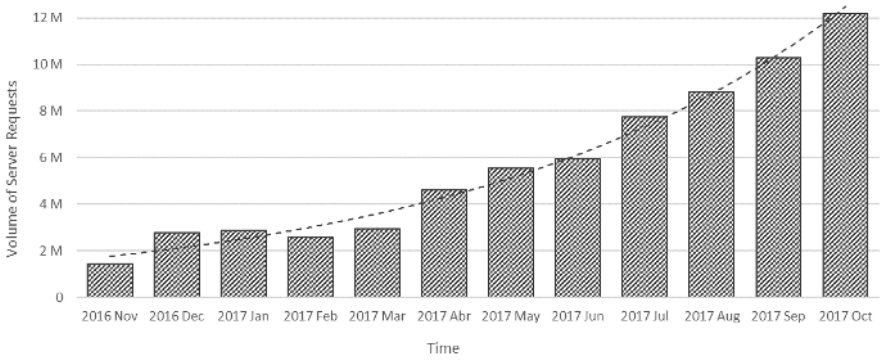


Figure 4.2 - Data Collection Growth Over Time

To support the SOC team, the scope of this work aims to develop a scalable Machine Learning implementation responsible for labelling the log events collected by the SIEM from the Apache Web Servers for a particular financial service, capable of handling the increasing data volume size.

The data collection process for this work starts with the creation of an exporter script that performs a request of the log files to the SIEM for a specific period of time. For every request, a collection of already structured data log files is output by SIEM in a single CSV format file, creating a pool of CSV files to generate what is the dataset of this work.

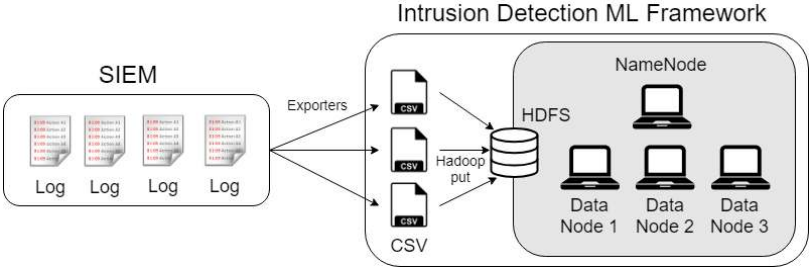


Figure 4.3 - Data Collection Flow from SIEM Logs to CSV to HDFS

Having the HDFS service running, the **CSV files** were ingested into **HDFS** using the available user commands from the running hdfs script, as displayed in Figure 4.3 (Apache Software Foundation, 2015). The HDFS through the NameNode, orchestrates the splitting of each file into 64MB blocks, writes and replicates them across the available DataNodes, manages and keeps track of the block mapping by storing its metadata. The end result are CSV files split into blocks and distributed across the worker nodes of the cluster, creating the distributed, highly available and resilient storage of the cluster, ready to be used by other services.

From the CSV in HDFS, two additional data storage formats were used, the **Apache Optimized Row Columnar format** (ORC) (Apache Software Foundation, 2018c) and **Apache Parquet** (Apache Software Foundation, 2018d), in order to provide grounds for a comparative analysis of the effects of different storage formats over the preprocessing and Machine Learning processing times in chapter 5. While the CSV file format provides a row-oriented file format, both ORC and Parquet provide a columnar-oriented file format with expected considerable storage reductions and processing times when compared with the CSV. Additionally, the two available variations for data storage compression were also used, the ZLIB and SNAPPY for ORC, and the GZIP and SNAPPY for the Parquet, as well as their respective uncompressed formats.

To produce them, and according to Figure 4.4, the CSV data in HDFS is loaded into Hive as an external table (1). Next, an empty Hive managed table with an equal schema is created, but with the storage pointing to ORC or Parquet formats with their respective compressions (2). The final step foresees the ORC/Parquet table to be overwritten by the external to be populated with the desired data in the correct storage format (3) (Hortonworks, 2018).

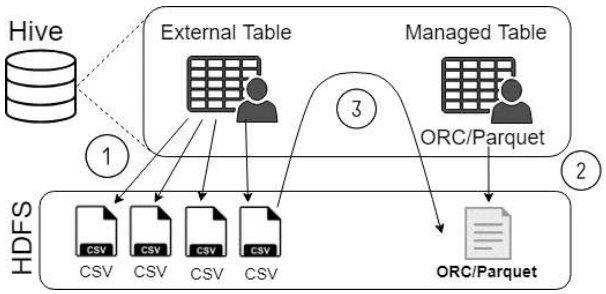


Figure 4.4 - Data Collection Flow from CSV to ORC and Parquet

From the previous descriptions, Table 4.2 summarizes the seven data storages used in this work:

Table 4.2 - Experimental File Formats and Compressions Used

Storage Format	CSV (HDFS)	ORC (HDFS)	ORC (HDFS)	ORC (HDFS)	Parquet (HDFS)	Parquet (HDFS)	Parquet (HDFS)
Storage Structures	Row-oriented	Column-oriented	Column-oriented	Column-oriented	Column-oriented	Column-oriented	Column-oriented
Compression Type	-	Native	ZLIB	SNAPPY	Native	GZIP	SNAPPY
Compressed	No	No	Yes	Yes	No	Yes	Yes

4.4.2. Data Description – Original Dataset

The data provided by the financial organization is a dataset composed of more than 73.7 million server requests of the financial service and 62 features in a dataset with over 8GB worth of storage. From these features, 54 are of the type categorical and the remaining 8 are interval types.

The features contained in the original dataset were analyzed and discussed with the Security team in order to remove redundant or uninformative data right from the start. This process resulted in the removal of 49 features while keeping 13 features for further analysis. All the column names have been changed from the original due to confidentiality reasons.

The following Figure 4.5 presents a small sample of the remaining available features:

id	timestamp	sourceAddressObf	deviceEventClassId	requestMethod	requestUrlFileName	requestClientDevice	bytes	referrer	depvar
0	2017/07/21 23:55:19 WEST	62381a8625	200	GET	cmpservice.com/lb_test.html	KeepAliveClient	2	null	0
1	2017/07/21 23:55:32 WEST	6723965073	200	POST	cmpservice.com/ndd/kap	Android	39	null	0
2	2017/07/21 23:55:31 WEST	437768569b	200	POST	cmpservice.com/ndd/kap	Android	12763	null	0
3	2017/07/21 23:55:32 WEST	18a790694c	200	POST	cmpservice.com/ndd/kap	Android	119	null	0
4	2017/07/21 23:55:32 WEST	98caeb140e	200	GET	cmpservice.com/lb_test.html	KeepAliveClient	5	null	0
5	2017/07/21 23:55:32 WEST	6fa79c7797	200	POST	cmpservice.com/ndd/kap	Android	31	null	0
6	2017/07/21 23:55:32 WEST	2cef4497b5	200	POST	cmpservice.com/ndd/kap	Android	41	null	0
7	2017/07/21 23:55:31 WEST	7a22033e48	200	POST	cmpservice.com/ndd/kap	iOS	15631	null	0
8	2017/07/21 23:55:33 WEST	6723965073	200	POST	cmpservice.com/ndd/kap	Android	350	null	0
9	2017/07/21 23:55:33 WEST	f9449baa1a	200	POST	cmpservice.com/ndd/kap	Android	40	null	0

only showing top 10 rows

id	timestamp	GeoCountryFlagUrl	destinationHostName	destinationAddressObf	transportProtocol	depvar
0	2017/07/21 23:55:19 WEST	Geo_flag_12	server2	1x4702a777	TCP	0
1	2017/07/21 23:55:32 WEST	Geo_flag_12	server2	1x4702a777	TCP	0
2	2017/07/21 23:55:31 WEST	Geo_flag_1	server1	933da0272b	TCP	0
3	2017/07/21 23:55:32 WEST	Geo_flag_12	server1	933da0272b	TCP	0
4	2017/07/21 23:55:32 WEST	Geo_flag_10	server2	1x4702a777	TCP	0
5	2017/07/21 23:55:32 WEST	Geo_flag_12	server2	1x4702a777	TCP	0
6	2017/07/21 23:55:32 WEST	Geo_flag_1	server2	1x4702a777	TCP	0
7	2017/07/21 23:55:31 WEST	Geo_flag_12	server1	933da0272b	TCP	0
8	2017/07/21 23:55:33 WEST	Geo_flag_12	server2	1x4702a777	TCP	0
9	2017/07/21 23:55:33 WEST	Geo_flag_1	server2	1x4702a777	TCP	0

only showing top 10 rows

Figure 4.5 - Original Dataset Sample

Table 4.3 displays a summary of their entity groups, feature names, roles and data types for the remaining features. From the table, the features are organized in seven different entities: i) Identification; ii) Time; iii) Network – Source Host; iv) Network – Destination Host; v) Application; vi) Request; and vi) Label.

The first two groups, the **Identification** and the **Time**, are related with the unique identification number assigned to the server request and the timestamp registered in the logging. The second two groups, the **Network Source and Destination Hosts**, contain the information regarding the identification of the source and destination instances involved on both ends of the request, in terms of source and destination IP, source country identification and internal server identification assigned to receive the request. The fifth group, the **Application**, is reserved for the information regarding the system used by the devices of the source instance to emit the financial service request to the server. A typical example used by mobile devices is the Android and iOS. It is important to refer that raw information contained additional data such as the physical mobile device brand and model, OS versions, among other information, that was removed for

confidentiality purposes by the SOC. The sixth group, the **Request and Response**, is the entity that gathers more features. The group holds data regarding the server request and response, containing information with the volume of bytes from the server response, request information from forth Open Systems Interconnection model layer (OSI layer) ("The OSI Model - Features, Principles and Layers," 2018), the transport level, with the transport protocol used (ex: TCP), and from the seventh OSI layer ("The OSI Model - Features, Principles and Layers," 2018), the application level, with information regarding the request method used (ex: POST, GET, etc), URL requested, and HTTP status code response from the request.

The last entity is the **Label**, containing the target variable which separates the normal service requests from intrusion attempts. In order to frame this work's cyber attacks types with other similar-natured works, and given the network attacks taxonomy (Hoque, Bhuyan, Baishya, Bhattacharyya, & Kalita, 2014) previously described on chapter 2, according to the SOC team, the financial service infrastructure is exposed to attacks of the type: i) **Application Layer Attack**, where all types of attacks can happen, depending on the application being attacked, which can damage the company that is providing the service or even the visitors; ii) **User Attacks**, where R2L attacks can take place; iii) active **Information Gathering** with direct server requests seeking to validate weak entries; and iv) **Malware**, the software itself that performs the malicious activity.

Table 4.3 - Original Features Names, Description, Role, and Data Type

Entity	Feature	Role	Data Type	Description
Identification	id	ID	Nominal	Server request unique ID
Time	timestamp	Input	Ordinal	Server request logged timestamp
Network - Source Host	sourceAddressObf	Input	Categorical	Source request IP obfuscated by the SOC
	GeoCountryFlagUrl	Input	Categorical	Source request country flag
Network - Destination Host	destinationHostName	Input	Categorical	Internal Server assigned by the load balancer to receive the service request
	destinationAddressObf	Input	Categorical	Destination IP obfuscated by the SOC
Application	requestClientDevice	Input	Categorical	Device used by the source to make the request
Request and Response	transportProtocol	Input	Categorical	Transport Protocol used for the request
	referrer	Input	Categorical	Identification of the URL that linked to the resource being requested. By checking the referrer, the new webpage can see where the request originated
	requestMethod	Input	Categorical	Request method used (eg. POST)
	requestUrlFileName	Input	Categorical	URL of the service request
	deviceEventClassId	Input	Categorical	HTTP status code response for the request
	bytes	Input	Interval	Volume of bytes from the server response
Label	depvar	Target	Binary	Binary Target variable with normal service requests (label 0) and intrusion attempt server requests (label 1)

4.4.3. Data Exploration – Original Dataset

A statistical overview analysis of the features available on the dataset in its original state was performed in order to acquired insights about the data. No further statistics of the original dataset are provided on the appendixes due to confidentiality reasons. It is important to note that only the features that were not excluded due to data quality and consistency reasons approached on the following subchapter are included in this statistical analysis. A **univariate exploratory analysis** of the **input nominal features** and some descriptive statistics were performed and summarized in Table 4.4.

Table 4.4 - Univariate Exploratory Analysis of the Input Nominal Features – Original Dataset

Features	Label	Number of Classes	Non-Missing Values Rows	Mode	2nd Mode
sourceAddressObf	0	(*)	73164516 (100%)	62381a8625 (<0.01%)	472bb54c78(<0.01%)
	1	(*)	588716 (100%)	c175b9c0f1 (<0.01%)	173ccf37be (<0.01%)
GeoCountryFlagUrl	0	37	73164516(100%)	Geo_flag_12 (*)	Geo_flag_23 (*)
	1	12	588716 (100%)	Geo_flag_23 (*)	Geo_flag_7 (*)
destinationHostName	0	2	73164516(100%)	Server1 (49.97%)	Server2 (50.03%)
	1	2	588716 (100%)	Server1 (55.46%)	Server2 (44.54%)
destinationAddressObf	0	2	73164516(100%)	933da0272b (49.97%)	1x4702a777 50.03%)
	1	2	588716 (100%)	933da0272b (55.46%)	1x4702a777 44.54%)
requestClientDevice	0	204	70545635 (96.42%)	Android (50.44%)	iOS (30.60%)
	1	55	47638 (8.09%)	Null (91,91 %)	Desktop (5.05%)
transportProtocol	0	1	73164516(100%)	TCP (100.00%)	-
	1	1	588716 (100%)	TCP (100.00%)	-
referrer	0	37	361 (<0.01%)	Null (>99.99%)	URL_103 (2.19%)
	1	124	14694 (2.50%)	Null (97.50%)	URL_443(<0.01%)
requestMethod	0	14	73164516 (100%)	POST (85.35%)	GET (14.60%)
	1	116	588716 (100%)	GET (96.88%)	POST (2.14%)
requestUrlFileName	0	503	73164516 (100%)	cmpservice.com/ndd/kap (83.84%)	cmpservice.com/lb_test.html (14.44%)
	1	11070	588716 (100%)	cmpservice.com/ (3.80%)	cmpservice.com/q79w_38jg__.shtml (0.85%)
deviceEventClassId	0	9	73164516(100%)	200 (99.57%)	403 (0.33%)
	1	9	588716 (100%)	404 (62.81%)	403 (32.75%)
(*) - Confidential					

Analyzing the statistics, regarding the univariate analysis of the input nominal features of the original dataset, it is possible to identify four out of five features as holders of high cardinality class levels, with special emphasis on the requestUrlFileName with up to 11070 classes solely for the label 1 and 503 for the label 0. The preprocessing of this high cardinality features stood as one of the most demanding tasks of this work and is addressed on the data preparation subchapter.

Both features requestClientDevice and referrer present events with missing values, being especially accentuated on the later one with only less than 3.00% of the events being identified as non-missing for both labels, as it was mentioned previously.

Although many of the features present high cardinality in terms of class levels, and not accounting for the missing values volume contribution, some of them present a disproportionate representation. This occurrence is especially noted on the requestMethod feature, where for the label 0 and 1 respectively, POST and GET methods contribute with over 85% and 96% of the total number of events. However, the most notable scenario stands for the deviceEventClassId with a total contribution of HTTP codes 200 accounting for 99.57% of the label 0 class representation. Taking into account the business context and the expected usage of the service (label 0), the classes associated with POST and GET methods, successful HTTP code responses 200 and repeated service URL requests are representative of regular usage of the financial service. As for the events associated with intrusion attempts (label 1), the GET method and HTTP code responses of 404 and 403 in abundance regarding client errors, such as the ones produced from an active vulnerability search of the service and requesting a non-existing content in their attempts, are unsurprising.

Table 4.5 - Univariate Exploratory Analysis of the Input Interval Feature – Original Dataset

Feature	Label	Non-Missing Values Rows	Mean	Standard Deviation	Min	Q25	Q50	Q75	Max
bytes	0	62196106(99.94%)	6652	47311	2	39	121	501	5354606
	1	586641(99.65%)	222	26	2	210	217	229	1490

For **univariate exploratory** analysis of the **input interval features**, some descriptive statistics are summarized on Table 4.5 (figure with a plot in Appendix 9.6). Analyzing the table regarding the only feature, the "bytes", of the original dataset it is possible to identify two distinct patterns regarding both target variables. On the intrusion attempts server requests (label 1), the frequency distribution presents values for mean and median quite close to each another, respectively with 222 and 217 bytes, and a low standard deviation value of 26 bytes when compared with the mean. However, on the normal service server requests (label 0), the frequency distribution of events displays a skewed representation with the mean and median values deviating highly form each other, respectively 6652 and 121 bytes, accompanied by standard deviation value seven times bigger than the mean value with 47311 bytes.

Considering the business context and the expected usage of the service (label 0), the minimum and maximum values noted for the feature Bytes are considered by the SOC as normal regarding the expected server requests of the financial service. As for the events associated with intrusion attempts (label 1), the low values associated when compared with the previous sentence are not shocking. The high number of

client error requests associated with HTTP codes 400s from the previous analysis indicates that the server requests were not attended and therefore the content of the request was not retrieved as intended.

Table 4.6 - Univariate Exploratory Analysis of the Target Variable

Label	Row Count	Contribution
0	73164516	99,20%
1	588716	0.80%
Total	73753232	100%

For the **univariate exploratory analysis** of the **target feature**, named "depvar", some descriptive statistics are summarized in Table 4.6. Analyzing the results, the target variable is binary and presents two classes, the one associated with normal server requests of the financial service (label 0) and the intrusion attempts server requests (label 1). The class related to the threat server request (label 1), contributes less than 1% of the whole dataset. According to Akbani, Kwek, and Japkowicz (2004), this uneven class distribution is expected in domains such as fraud detection, which is similar to this work's domain, where ratios of 100 to 1 or even 100000 to 1 are recurrently present. Following the examples given by Hakim, Sartono, and Saefuddin (2017), class imbalance problems are present in class distribution ratios of 100 to 1 onwards. Therefore, it is concluded that our dataset falls under the experience of what is considered as an imbalanced dataset. The consequences, approaches and handling of this work's imbalanced dataset were addressed in the modelling chapter.

4.4.4. Data Quality and Consistency Validation – Original Dataset

During the data exploration stage, data consistency and quality validations were conducted in order to identify issues that would require intervention during the preprocessing stage. The presence of missing values, outliers, invalid, redundant or obsolete categories, duplicated information, confusing semantics, inconsistent data, inadequate data, among other issues, are regular problems present in most real-life business activity databases.

Table 4.7 – Data Quality Validation for Missing Values

Feature	Label	Missing Values Rows	Percentage by label	Percentage Total
requestClientDevice	0	2610786	3.57%	3.54%
	1	541078	91,91%	0.73%
referrer	0	62235537	99.99%	99.06%
	1	574022	97.50%	0.91%
bytes	0	39772	0.06%	0.06%
	1	2075	0.35%	0.003%

For the dataset available, the issues identified are mainly associated with: i) missing values; ii) inadequate data for the problem we are aiming to solve; and iii) uninformative or redundant data. The first type of issue, the **missing values**, was identified in three features: "requestClientDevice", "referrer" and "bytes", as displayed on Table 4.7. Observing the table, the features "requestClientDevice" and "referrer" are particularly affected by the presence of missing values. Both features have more than 90% of the label 1 rows with no content. Particularly the feature "referrer" has the majority of its content with missing values, more than 99% on the total dataset. According to the security team, for both "requestClientDevice" and "referrer" features is expected the presence of missing values. However, for the feature "bytes" the presence of missing values was not expected by the security team. The presence even if in a low percentage of missing values, below the 0.5% for both labels and below 0.1% on the total dataset, should be handled.

The second issue identified, the **inadequate data**, is related to log server requests from the financial organization's internal usage of the services (identified by specific sourceAddressObf and requestClientDevice values, undisclosed in this work due to confidentiality reasons) and therefore don't fit the purposes of this work, as they are not from clients.

Table 4.8 - Data Quality Validation for Inadequate Data

Source of Server requests	Row Count	Percentage
Internal usage server requests	10928638	14.82%
Client usage server requests	62824594	85.18%
Total	73753232	100.00%

As it can be observed on Table 4.8, a total of more than 10.9 million rows (all non-threat server requests), corresponding to 14,82% of the dataset, have been identified as unsuited to later feed our models and was address on the data preprocessing stages of this work.

Another problem associated with inadequate data is related to the content of some of the features. The attributes associated with network source host, "sourceAddressObf", "GeoCountryFlagURL", and the "timestamp" should not be used for our models due to the methodology used to generate the target variable. Most of the events labelled as intrusion attempts were identified through the IPs used by the company that performs the periodic vulnerability assessment. As stated previously, most tests are performed in batch once or twice a week and from a limited range of IPs. As such, in order to avoid biased or unrealistic information towards the detection of a real intrusion, it was decided to remove them even if it meant losing potentially important information about the client that makes the request.

The third issue identified, the **uninformative or redundant data** is related to features that have the same value in every row of the dataset. Even though the content of these features provides additional information for the business context, their lack of variance as training data to feed our models is not

beneficial. Thus, the feature associated with the transport protocol should be discarded due to their lack of information potential. The "transportProtocol" feature presented in every server request the same information: "TCP/IP".

The features associated with the network destination host (destinationHostName and destinationAddressObf), should be discarded due to their uninformative information, accordingly with the organization's security expertise. The content of their information is related to the internal server that receives the client request and the load balancer. Since every server has the same configuration and the source of intrusion intended to be detected is not DoS/DDoS, the potential information in every server request is the same.

4.5. DATA PREPARATION

The data preparation stage comprises every action taken over the original dataset to create a dataset that ultimately will be used on the modelling stage. The following groups of actions were taken: i) Data cleaning; and ii) Data transformation, where it is included the feature engineering and the feature selection.

4.5.1. Data Cleaning

The data cleaning stage is focused on handling the issues identified during the data exploration phase. Two measures were taken to solve the issues related to the **inadequate data**: i) All the rows associated with Internal usage server requests were filtered out (10928638 rows, 14.82% of the dataset); and ii) The features "sourceAddressObf", "GeoCountryFlagURL", and the "timestamp" were discarded from the dataset that will feed the models.

To handle the **missing values**, three measures were taken:

- The missing values of the feature "requestClientDevice" were not filtered out. All the information missing was found valuable by the security team. A portion of the missing values when mapped with specific URL classes from the feature "requestUrlFileName" were named "merchant_other_requests" (520354 rows associated to label 0). The remaining values missing were assigned to a new class named "unknown" requests (2090432 rows associated to label 0 and the remaining 541078 rows to label 1);
- The missing values from the feature "referrer" were handled through the assignment of a new class "no_msg" (62235537 rows associated to label 0 and the remaining 574022 rows to label 1). The content of this feature is only filled if a message exists to be displayed. The lack of content in it provides us with the important information that there is no message to be displayed;
- The remaining missing values, belonging to the feature "bytes", were filled with the median values of the label 0 and 1.

The third identified issue, the **uninformative or redundant data**, was handled by discarding the features "transportProtocol", "destinationHostName" and "destinationAddressObf" as they would not provide any value for our classifiers.

4.5.2. Data Transformation

4.5.2.1. Feature Engineering

The data transformation stage seeks the creation of new features engineered from the original ones. One of the problems identified previously was the high number of categories in most categorical features that could potentially lead to problems related to the curse of dimensionality and overfitting of the classification models. This issue was addressed through binning of the categories in most features in order to reduce the number of unique values.

The feature "deviceEventClassId" ranged categories of HTTP status codes from 200 to 599 in a total of 18 categories. The following binning categories were used to create a new feature – "**httpCodeGrouped**": i) category "2xx" – Success response – Binned all the categories from 200 to 299; ii) category "3xx" - Redirection response – Binned all the categories from 300 to 399; iii) category "4xx" - Client errors response – Binned all the categories from 400 to 499; and iv) category "5xx" - Server errors response – Binned all the categories from 500 to 599;

The feature requestMethod contains 117 different methods in the whole dataset. However, accordingly with the Organization's Security Expertise, the financial service normal usage, from whom the logs come from, is expected to receive only requests using GET or POST methods. Therefore, the following binning categories were used to create a new feature – "**requestMethodGrouped"**: i) category "common_get" – Binned all the requests from GET methods; ii) category "common_post" – Binned all the requests from POST methods; and iii) category "uncommon" – Binned all the requests not expected methods.

The feature referrer contained 162 unique categories in the whole dataset. The majority of the rows previously empty were handled during the data cleaning step. Therefore, the following binning categories were used to create a new feature – "**referrerContentGrouped**": i) category "no_msg" – Rows containing "no_msg" category; and ii) category "with_msg" – Binned all the categories that did not contain "no_msg".

The feature requestClientDevice contained 252 different devices used by clients in the whole dataset. The guidance of the Organization's Security Expertise team led to the definition of the following binning categories to reduce the number of categories and create a new feature – "**requestClientDeviceGrouped**": i) category "android" – requests from android mobile devices; ii) category "desktop" – requests from desktop devices; iii) category "ios" – requests from iOS devices; iv) category "merchant_bot_requests" –

Fields that contain information with certificate configurations tests, bots and crawler related requests from merchant sources; v) category “merchant_programatic_requests” – Fields that contain information with java, python, Hypertext Preprocessor (PHP) and other programmatic requests from merchant sources; vi) category “merchant_other_requests” – Merchant service requests not contemplated on the bot or programmatic requests; vii) category “not_merchant_bot_requests” - Fields that contain information with certificates, bots and crawler related requests from non- merchant sources; viii) category “not_merchant_programatic_requests” - Fields that contain information with java, python, PHP and other programmatic requests from non-merchant sources; iv) category “tool_requests” – Fields that contain tools or methods to access contents of the requests such as “curl”, “wget” among others; x) category “unknown_requests” – Fields that contain the previously replaced information of “unknown”. In other words, requests that previously were with missing values and at the same time didn’t fit the merchant_other_requests bin; and xi) others - requests out of the range of the previous groups were binned here.

The feature requestUrlFileName contained 11529 different devices used by clients to make the server requests. The experience of the Organization’s Security Expertise team lead to the definition of the following binning categories to reduce the number of categories and create a new feature – “**requestUrlFileNameGrouped**”: i) category “urlFolder” – URL server requests to access folders and therefore with no extension; ii) category “urlFile_wExt_image” – URL server requests to access a file of type image; iii) category “urlFile_wExt_notImage” – URL server requests to access a file different from an image; and iv) category “urlFile_noExt” – URL server requests to access a files but without extension.

4.5.3. Data Exploration of the Transformed Data

A brief data exploration was performed over the final transformed data. Further statistics and are addresses on Appendix 9.7. The following Figure 4.6 displays a small sample of the transformed data:

dt_timestamp	dt_epoch	referrerContentGrouped	requestClientDeviceGrouped	requestMethodGrouped	httpCodeGrouped	requestUrlFileNameGrouped	bytes_t	depvar
2017-09-28 23:53:56	1506639236	no_msg	ios	common_post	2xx	urlFile_noExt	218	0
2017-09-28 23:53:56	1506639236	no_msg	unknown_requests	common_post	2xx	urlFile_noExt	487	0
2017-09-28 23:54:04	1506639244	no_msg	ios	common_post	2xx	urlFile_noExt	487	0
2017-09-28 23:54:04	1506639244	no_msg	android	common_post	2xx	urlFile_noExt	39	0
2017-09-28 23:54:04	1506639244	no_msg	ios	common_post	2xx	urlFile_noExt	39	0

only showing top 5 rows

dt_timestamp	dt_epoch	referrerContentGrouped	requestClientDeviceGrouped	requestMethodGrouped	httpCodeGrouped	requestUrlFileNameGrouped	bytes_t	depvar
2017-02-21 17:37:42	1487698662	no_msg	unknown_requests	common_get	3xx	urlFolder	229	1
2017-02-21 17:37:43	1487698663	no_msg	unknown_requests	common_get	3xx	urlFolder	229	1
2017-02-21 17:38:13	1487698693	no_msg	unknown_requests	uncommon	4xx	urlFile_wExt_notI...	212	1
2017-02-21 17:38:13	1487698693	no_msg	unknown_requests	common_get	3xx	urlFolder	229	1
2017-02-21 17:38:03	1487698683	no_msg	unknown_requests	common_get	3xx	urlFolder	229	1

only showing top 5 rows

Figure 4.6 - Transformed Dataset Sample

The following Table 4.9 presents a **univariate exploratory analysis** of the **transformed input nominal features** with some descriptive statistics:

Table 4.9 - Univariate Exploratory Analysis of the Input Nominal Features – Transformed Dataset

Features	Label	Number Classes	Mode	2nd Mode
httpCodeGrouped	0	4	2xx (99.66%)	4xx (0.30%)
	1	4	4xx (97.28%)	3xx (2.35%)
referrerContentGrouped	0	2	no_msg (99.99%)	with_msg (0.01%)
	1	2	no_msg (97.50%)	with_msg (2.50%)
requestMethodGrouped	0	3	common_post (99.99%)	common_get (0.06%)
	1	3	common_get(96.88%)	common_post (2.14 %)
requestUrlFileNameGrouped	0	4	urlFile_noExt (96.88%)	urlFolder (0.008%)
	1	4	urlFolder (48.46%)	urlFile_wExt_notImage(31.97%)
requestClientDeviceGrouped	0	11	Android (59.08%)	iOS (35.78%)
	1	5	unknown_requests(91.91%)	desktop (5.05%)

Analyzing the statistics regarding the univariate analysis of the transformed nominal features it is possible to identify that the binning process leads to a significant reduction of the high cardinality of the class levels of most features of the original dataset. After the preprocessing, the highest number of class levels belongs to the binned "requestClientDevice", now identified as "requestClientDeviceGrouped", with 11 and 5 class levels respectively for the target variable label 0 and 1, previously were respectively 204 and 55 class levels. All remaining transformed features have now between 2 to 4 class levels either for intrusion attempts (label 1) and normal service requests (label 0), including for the feature previously identified as "requestURLFileName", now named "requestUrlFileNameGrouped", were the highest cardinality was registered with 503 and 11070 levels respectively for the target variable label 0 and 1.

Another important observation is that, even though no missing values are present at this point, the same disproportionate class level representation is still present among the features. The same observations still stand as for the framing with the business context: i) For the expected normal usage of the service (label 0), the classes associated with POST method ("common_post" with a contribution of 99.99%)) from feature "requestMethodGrouped" (previously identified as requestMethod), and the successful HTTP code responses 200, now binned ("2xx") in the feature "httpCodeGrouped", are still to be expected; and ii) For the events associated with intrusion attempts (label 1), the GET method ("common_get" with a contribution of 96.88%) from feature "requestMethodGrouped" (previously identified as requestMethod) and HTTP code responses 404 and 403 now binned ("4xx" with a contribution of 97.28%) in the feature "httpCodeGrouped", regarding client errors, are still to be expected.

As for the remaining features, the "requestUrlFileNameGrouped" (previously known as "requestUrlFileName") denotes a higher presence of URLs requests with no extensions ("urlFile_noExt" with a contribution of 96.88%) for normal service requests (label 0), while the intrusion attempts (label 1)

have the major expression of requests on requests regarding folders (urlFolder with a contribution of 48.46%).

The feature "requestClientDeviceGrouped" identify more than 93% of the requests from normal service usage (label 0) from mobile devices from "Android" and "iOS", while a more than 91% of the number of intrusion attempts (label 1) are from devices not present in the organization's device dictionary and have been binned and renamed "unknown_requests".

The referrerContentGrouped, where mostly missing values were present in the past now renamed with "no_msg" with representations on both labels above 97%, presents a small portion of relevant messages in the intrusion attempts of 2.50%, that contrast with the 0.01% of messages present in the normal usage requests (label 0).

Table 4.10 - Univariate Exploratory Analysis of the Input Interval Feature – Transformed Dataset

Feature	Label	Non-Missing Values Rows	Mean	Standard Deviation	Min	Q25	Q50	Q75	Max
bytes_t	0	62235878	7776	51023	2	40	218	624	5354606
	1	588716	222	26	2	210	217	229	1490

Analyzing the statistics regarding the univariate analysis of the transformed numeric feature from Table 4.10, the "bytes_t", taking in account the business context, the only transformation applied was filtering out the few missing values that were present. Therefore, the previous observations regarding the features "bytes" still stand.

For the **univariate exploratory analysis** of the **target feature**, named "depvar", some descriptive statistics were performed and summarized in Table 4.11.

Table 4.11 - Univariate Exploratory Analysis of the Input Target Variable – Transformed Dataset

Label	Row Count	Contribution
0	62235878	99,06%
1	588716	0.94%
Total	**62824594**	**100%**

Analyzing the results of the previous table, after the preprocessing, the class related with the threat server request, identified by the label 1, is still contributing with less than 1% of the whole dataset. Therefore, the previous analysis related to imbalance dataset still stands.

4.5.1. Feature Selection

Feature selection is a standard procedure that aims to perform dimensionality reduction in order to increase the classifiers generalization performance while reducing the training and testing time (Basu & Murthy, 2012; Suh-Lee et al., 2016).

According to the available features, even if few, the majority of them are categorical-natured with not so low cardinality, where only one feature is of numeric type. What is represented by a few features can rapidly escalate to multiple features on learning algorithms that require methods like the creation of dummy variables to handle categorical features. Therefore, it is important to perform an analysis to assess if the features are worth not being discarded over its influence on the learning algorithm's performance to generalization tasks.

According to Kawakubo and Yoshida (2012) there are three types of variable selection approaches: i) "filter"; ii) "wrapper"; and iii) embedded". The first one, the **filter**, ranks and chooses a subset of features during the preprocessing stage without knowledge of the learning algorithm chosen. The second, the **wrapper**, ranks and chooses a subset of features according to its predictive power associated with a learning algorithm. The third approach, the **embedded**, performs feature selection during the learning process associated with specific learning algorithms.

In this work, the dataset is comprised of five categorical features and one numerical feature. While some of the traditional feature selection approaches through filtering highly correlated numeric features using, for example, the Pearson Correlation (Sisiaridis & Markowitch, 2017), cannot be applied in our context due to the existence of a single numeric feature, the same is not valid for the categorical features.

With the objective of exploring the Big Data technology of this work to the fullest, according to Sisiaridis and Markowitch (2017), two feature selection approaches can be used with the Spark Framework: i) Chi-Square Test, as a filter approach; and ii) Random Forests, as a wrapper approach (Kawakubo & Yoshida, 2012).

The first one, the **Chi-Square Test**, is a popular statistical test that measures the independence of two events. As a feature selection method, it can be used to measure the independence of each individual categorical or nominal feature over the class labels of the target variable (Cambridge University Press, 2008b; Jamali, Bazmara, & Jafari, 2012). Each feature is ranked according to the following quantity formula eq.1 (Cambridge University Press, 2008b):

$$\chi^2(D,t,c) = \sum_{e_t \epsilon \{0,1\}} \sum_{e_c \epsilon \{0,1\}} \frac{(N_{e_t e_c} - E_{e_t e_c})^2}{E_{e_t e_c}} \quad \text{(eq.1)}$$

Where N stands as the observed frequency in D, and E stands as the expected frequency of t and class c occurring together assuming that they are independent.

The chi-square test measures how much expected events counts and observed events counts deviate from each other by testing the hypothesis of independence between them (Cambridge University Press, 2008b; Forman, 2003). According to the Cambridge University Press (2008), the rationale of using the test as a feature selection method stands as follows: If the χ^2 value is higher than a defined threshold of probability of occurrence, than the outcome is statistically significant for the associated probability p-value of occurrence and the hypothesis of independence is rejected, thus deeming the feature informative enough for being worth keeping for predicting the target variable (Cambridge University Press, 2008b). In the other hand if the hypothesis is not rejected than the independency test is verified and the feature is not relevant enough as a potentially worth keep feature for the prediction of the target variable (Cambridge University Press, 2008b).

The dataframe API used by the spark.ml package offers the class ChiSquareTest and ChiSqSelector with an implementation of the chi-square test and feature selection of categorical attributes regarding the prediction of a categorical label, (Apache Software Foundation, 2018g).

The second method from the Spark Framework, the **Random Forests (RF)**, is an ensemble tree-based learning algorithm that has been widely explored and used as practical wrapper method for feature selection (Kawakubo & Yoshida, 2012).

The algorithm incorporates the concept of information gain as part of the decision-making process during the training phase. During the training phase of each decision tree, the node-splitting process occurs as a result of the calculation of the homogeneity measure of node, also known as impurity metric (e.g: for classification problems the Entropy or the Gini impurity) (Apache Spark, n.d.-c; Chen, Li, Member, Tang, & Bilal, 2017). For each potential feature split, the information gain is calculated as the difference between the parent node impurity and the weighted sum impurities of the potential child nodes. The learning algorithm will split the node on the feature that maximizes the information gain in each partition (Apache Spark, n.d.-c; Chen et al., 2017).

From the resulting fully-grown tree and node splits, the feature importance can be calculated, ranking the features by their contribution over the tree growth. Among the different implementations of the algorithm for ranking features according to its importance, the Gini Importance (GI), also known as Mean Decrease in Impurity (MDI), stands as one of most used (Kawakubo & Yoshida, 2012) and is the method implemented in the Spark framework used in this work. The estimation of the feature importance implemented by the

Spark.ml package follows the work of Leo Breiman and Adele Cutler over Random Forests through the process

The "importance of a variable X_m for predicting Y by adding up the weighted impurity decreases $p(t) \cdot \Delta_i(s_t, t)$ for all nodes t where X_m is used, averaged over all N_t trees in the forest" (Louppe, Wehenkel, Sutera, & Geurts, 2013) and "and where p(t) is the proportion N_t/N of samples reaching t and $v(s_t)$ is the variable used in split s_t." (Louppe et al., 2013) is given by equation eq.2:

$$Imp(X_m) = \frac{1}{N_T} \sum_T \sum_{t \in T: v(s_t) = X_m} p(t) \cdot \Delta_i(s_t, t) \quad \text{(eq.2)}$$

In other words, the average sum of gain over all nodes which split over the feature X_m across all trees, weighted by the number of instances it splits by passing through the node (Apache Spark, n.d.-c).

The spark.ml package offers a class called RandomForestClassifier with an implementation of the Random Forests algorithm with the option of calling the feature importance through the MDI approach after training the class (Apache Spark, n.d.-c).

For this work, both feature selection methods were used but in different parts of the whole pipeline. The first one, the Spearman's Chi-Square Test of Independency was applied for the categorical data during the data preparation stage over the preprocessed features of the dataset with the objective of filtering out potential irrelevant features for the prediction of the target variable. As for the second method described, the usage of the Random Forests as a wrapper method was applied during the training of the learning algorithm. The results are presented analyzed and discussed in chapter 5 as one of the performance analysis of this work. Therefore, for each of the transformed categorical features, the Spearman's Chi-Square test of independence was performed over the target feature "depvar". The following Table 4.12 summarizes the results:

Table 4.12 - Feature Selection Using Chi-square as a Filter

Feature	Number of Levels	Degrees of Freedom	χ^2	p-value
requestMethodGrouped	3	2	59542556	< 0.001
requestUrlFileNameGrouped	4	3	50325345	< 0.001
httpCodeGrouped	4	3	46993624	< 0.001
requestClientDeviceGrouped	10	9	14392527	< 0.001
referrerContentGrouped	2	1	1508634	< 0.001

Analyzing the previous Table 4.12, for each feature and their respective number of degrees of freedom, a p-value lower than 0.001 is outputted from the independency test. These results mean that for each of the feature tested, there is a strong presumption against the null hypothesis, i.e., the independency between

the feature and the target variable to be statistically significant. Therefore, the null hypothesis is rejected as the features are relevant for predicting target variable and none of the features is filtered out.

4.6. MODELLING

The data modelling stage proceeds the dataset preprocessing and comprises the definition and reasoning behind the modelling techniques applied, and respective parameters used to perform. Starting with the definition of the basis of our modelling problem, a binary classification, this subchapter is comprised by the following parts: i) the data partition for definition of the training, validation and testing sets; ii) handling of the dataset imbalance problem; iii) hyperparameter tuning and training overfit control; iv) distributed processing modelling through the Spark ML modelling; and v) definition of all the learning algorithms used to perform classification tasks, along with all the relevant parameters used.

4.6.1. Binary Classification

According to Murphy (2012), the goal of a classification problem is to learn a mapping from a set of input features x, also referred as predictors, to an output categorical target variable y∈,1, … , C., where C represents the number of classes. For problems where C=2, the labels are mutually exclusive and the modelling is referred as to **binary classification**, often represented as $y \in \{0, 1\}$.

Given a labelled training set $D_N = \{(x_1, y_1), \dots, (x_N, y_N)\}$ comprised by N observations, with their respective predictors x and label y, and assuming $y = f(x)$ as an unknown function, the objective of learning using the training set is to produce an estimation of the function $f(x)$. From the estimation, prediction tasks over new unlabeled inputs can be performed, using $\hat{y} = \hat{f}(x)$, in what is referred as the model's ability to generalize (Murphy, 2012).

4.6.2. Data Partition

This step has the objective of describing the implemented sampling techniques to partition the dataset into train, validate and test datasets used on the classifiers, as well as handling the issues related to class imbalance. The summarized process of sampling is represented in Figure 4.7:

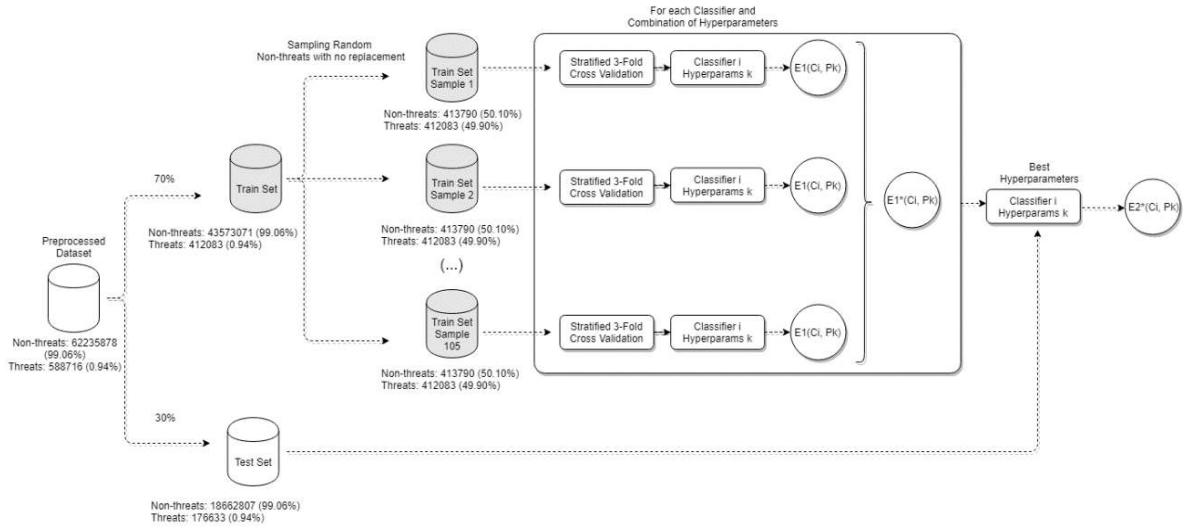


Figure 4.7 - Data Partition Architecture

The classification task involves two separate datasets (Cieslak, Chawla, & Striegel, 2006): i) train set; and ii) test set. In order to ensure control over the representation of both classes, the preprocessed dataset sampling started with the stratified partition of the initial dataset through random splitting into train and test datasets, respectively in 70% and 30% portions, in what is referenced as the holdout method (Kohavi, 1995).

According to (Kohavi, 1995), the **Holdout method**, also called the test sample estimation, partitions the data into two independent subsets called a training set and a test set. The partitions are used to train an estimator and assess its performance against unseen data. The method is considered a pessimistic estimator as it only uses a fraction of the data to assess its performance (Kohavi, 1995). The higher the number of instances used for testing the more realistic is this work's performance assessment of the learner, and the wider the range of instances used for learning of the estimator the higher the performance against unseen instances (Kohavi, 1995). The method creates an independent train and test sets without increasing substantially the computational process when compared to other methods. However, according to Kohavi (1995), assuming a finite and reduced dataset size, the method makes inefficient use of the dataset intrinsic information available by not using a significant portion for training purposes (2/3 to 70%). Given the volume of information available for our work the impact of the shortcoming of implementing this method is considered reduced.

For this work, the Table 4.13 summarizes the initial dataset partition from the holdout method, prior to the class imbalance handling. For the preprocessed dataset of this work, after partitioning, the test dataset was kept with its natural imbalanced class representation, with a majority class of 99.06% (class 0, non-threats) and the remaining 0.96% for the minority class (class 1, threats). However, for the train dataset additional

steps were taken to handle the shortcomings of low performant classifiers on predicting the minority class when handling an imbalanced dataset.

Table 4.13 - Training and Test Set Data Partition

Dataset	Partition	Number of Rows	Class Representation
Training Set	70%	43985154	Label 0 - 43573071 (99.06%) Label 1 - 412083 (0.94%)
Test Set	30%	18839440	Label 0 - 18662807 (99.06%) Label 1 - 176633 (0.94%)

4.6.2.1. Imbalanced Dataset Handling

The problems associated with training classifiers, using datasets with highly uneven of class distributions of the target variable, have been approached from different perspectives in several works (Bhowan, Johnston, & Zhang, 2012; Hakim et al., 2017; Singh & Purohit, 2015). In binary classification problems this means that one of the classes is under-represented, usually referred to as minority class, while the other class is referred to as the majority class (Singh & Purohit, 2015). According to Singh & Purohit (2015), training classification algorithms with imbalanced class distribution will lead to performance bias, where the majority class results will achieve high performances, contrasting the poor performance achieved for the minority class.

According to Galar, Fernández, Barrenechea, Bustince, and Herrera (2012), researches over the years to handle class imbalance problems can generally be categorized as: i) External or data level approaches; ii) Internal or algorithm level approaches; and iii) cost-sensitive learning approaches. The first category, the **external or data level approaches**, are methods focused on reshaping or sampling the original imbalanced dataset in order to produce a balanced dataset for the training, while leaving the learning algorithm unchanged (Singh & Purohit, 2015). Among studies, three approach groups of methods are documented (Li, 2007): i) Sampling methods; ii) Bagging-based methods; and iii) Boosting-based methods. The second category, the **Internal or algorithm level**, approaches are a group of methods focused on modifying the learning algorithm to accommodate the imbalance of classes, leaving the training data unbalanced as it is (Li, 2007; Singh & Purohit, 2015). The general approach of this methods revolves around implementations on the learning algorithms to bias the learning to account for the minority class (Akbani et al., 2004; Galar et al., 2012; Hakim et al., 2017; Li, 2007). The third category, the **cost-sensitive**, are methods, are described by Galar et al. (2012) and Hakim et al. (2017) as a midground between the data level and algorithm level approaches. These methods are generally focused on the attributing misclassification costs to the instances of each class and changing the algorithm's learning process to accept the introduced penalties, in order to bias the classifier's learning more aware of the minority class (Galar et al., 2012).

From the previous, the sampling and bagging-based methods, from the external approaches, are the most relevant for this work. The **sampling methods** provides techniques that seek to change the training set in order to balance it for the learning algorithm, such as the reduction of the majority class or increase of minority class, through undersampling and oversampling respectively (Singh & Purohit, 2015). The second approach, the **bagging-based methods**, belong to the group of ensemble methods where sampling techniques are repeatedly applied with replacement on the original imbalanced dataset to produce multiple balanced datasets (Galar et al., 2012; Hakim et al., 2017). An extended description of the external approaches (sampling, bagging-based, boosting-based methods), is presented on Appendix 9.8.

A series of variations and combinations of the previously presented approaches can be found throughout the literature across different fields of implementation. Overviewing the approaches, "in general, algorithm level and cost-sensitive approaches are more dependent on the problem, whereas data level and ensemble learning methods are more versatile since they can be used independently of the base classifier." (Galar et al., 2012). From the several sampling techniques capable of reshaping the dataset distribution to handle the imbalance issues, the Bagging Ensemble Variation (BEV) was used.

The **BEV** combines the concepts of undersampling of the majority class with the classical bagging. The implementation of the method starts by sampling without replacement the majority class data into $N = N_{Majority\ Class}/N_{Minority\ Class}$ sets, where $N_{Majority\ Class}$ is the number of instances from the majority class and $N_{Minority\ Class}$ is the number of instances from the minority class. For each sampled set from the majority class, all the minority class instances are added to create a subset of the original training set with an equal balance between classes. Each of the subsets is then used to train a classifier and the results ensembled through majority voting to output a final classification. The process is summarized on Figure 4.8.

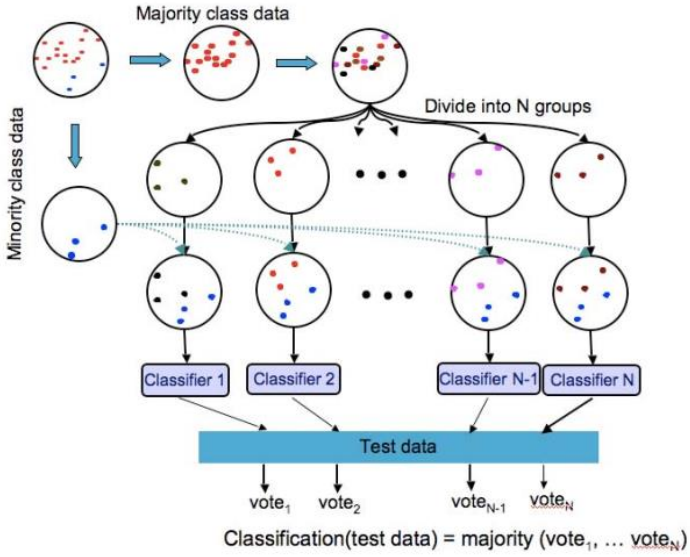


Figure 4.8 - The BEV System for Classifying Imbalanced (Li, 2007)

This approach allows the mitigation of one of the downsides of undersampling the majority class, losing potential important information, by ensuring that all majority class instances contribute for classifiers ensembled outcome, while using minority class instances without creating synthetic data (Li, 2007).

For this work, the following Table 4.14 summarizes the training and test sets used, after the data imbalance handling with BEV:

Table 4.14 - Training an Test Set After BEV Implementation

Dataset	Number of Subsets	Number of Rows (Each Subset)	Number of Rows (Total)	Class Representation (Each subset)
Training Sets	105	825873	86716665	Label 0 - 413790 (50.10%) Label 1 - 412083 (49.90%)
Test Set	1	18839440	18839440	Label 0 - 18662807 (99.06%) Label 1 - 176633 (0.94%)

According with Table 4.14, and assuming the previously presented contributions of each class on the preprocessed dataset, a total of 105 training subsets were sampled, each with 412083 instances associated to class 1 (threats) and 413790 instances of the class 0 (non-threats), with a balanced contribution of 50.10% ratio between classes, in favor of the non-threat class.

4.6.2.2. Hyperparameter Tuning and Training Overfit Control

For each of classifiers used over each of the created training subsets, the algorithms were trained and the hyperparameters tuned using the method of k-Fold CV.

According to Mitchell (1997), providing a validation set to the algorithm in addition to the training data is one of the most effective methods of overcoming the overfit of the model during the training. The performance of the learner is assessed through the validation set while increasing its classification complexity over each training instance used. The best set of hyperparameters are the ones most fit over the algorithm's performance against the validation set.

The **k-fold Cross Validation**, also known as rotation estimation, partitions the training set into k disjoint subsets, each with size m/k, being m the total number of instances available for training (Mitchell, 1997). The class ratio among partitions is commonly preserved through a rearrangement of data process called stratification (Nadiammai & Hemalatha, 2012). The learning algorithm is trained and assessed k different times, each using a different partition for validation and the remaining k-1 partitions for training (Mitchell, 1997; Nadiammai & Hemalatha, 2012). The results for all the k runs are averaged to produce an overall performance evaluation of the training, allowing all the instances to be used both for training and assessment (Mitchell, 1997). The greater the number of k folds used the greater the computational performance.

In this work, given the dataset size (825873 observations in each of the 105 training subsets), a stratified 3-fold CV was performed to assess and select the most promising combination of hyperparameters for each classifier, without excessively increasing the computational load. The best set of hyperparameters for each training algorithm represent the best average hyperparameters across all the 105 training subsets created from the BEV sampling.

After training, the unbiased performance of the algorithm is achieved through the assessment of the classification algorithm with the chosen set of hyperparameters from the training over the unseen examples of the test set.

4.6.3. Spark ML Modelling

In order to create models capable of processing the volume of information present on datasets of this work, the Apache Spark Framework was chosen to provide a distributed processing modelling environment to our data applied to the Machine Learning paradigm. The Apache Spark Framework provides a python API library with the implementation of Machine Learning algorithms and content related applied to the dataframe abstraction in the "pyspark.ml.package" (Apache Spark, n.d.-c).

The creation of Machine Learning models using the Spark ML package picks up after the creation of the training sets and follows a pipeline of data preparation steps to shape the training set to a valid input format to train the classification algorithms and the classifiers training process. The process is summarized in Figure 4.9.

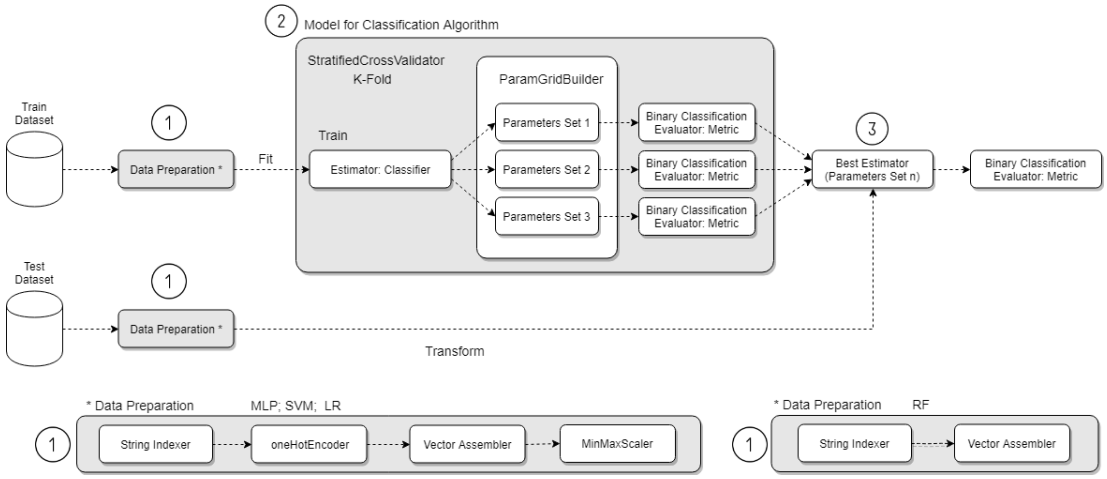


Figure 4.9 - Spark ML Modelling Architecture

Analyzing the components of Figure 4.9, two data preprocessing pipelines can be identified as a result of the different data preparations that each classification algorithm requires. Therefore, using the pyspark.ml.feature package, the first preprocessing pipeline is implemented for the Neural Networks,

Logistic Regression and SVM algorithms, and the second pipeline for the tree-based algorithms, in this work the Random Forests.

For the **first preprocessing pipeline** (1), all the categorical features need to be encoded and the interval features to be scaled. To achieve it, four transformations were implemented on spark: i) **String Indexer**, responsible for the encoding of each category on each categorical feature to a numeric integer format; ii) **One Hot Encoding**, responsible for creating dummy variables for each class of each feature; iii) **Vector Assembler**, responsible for grouping all features in a single column to create a dataframe composed by "features" and "labels" in order to create a required format to be used as input data for the algorithms classes; and iv) **MinMax Scaler**, a spark class that receives the vector assembled data and applies a Min-Max scale to the interval features.

For the **second preprocessing pipeline** (1), all the categorical features can and should be directly fed to the Random Forests algorithm and no interval features requires to be scaled. However, the spark implementation still requires the following transformations: i) **String Indexer**, even though the algorithm does not need the categorical features to be one hot encoded, they still require to be encoded, in other words, to be in a numeric format; and ii) **Vector Assembler**, responsible for creating a dataframe with the features grouped, as explained before. The resulting output of the pipelines is a vector format data object with all the information represented in two columns ("features" and "labels"). This is the format required as input data for the spark.ml classifiers objects.

The next modelling step (2), comprises the **implementation of the classification algorithms** using stratified k-fold cross validation sampling technique, model evaluation metric while performing parameter grid search to perform hyperparameter tuning. In order to do it, the StratifiedCrossValidator class from the spark_stratifier library was used (very similar with the CrossValidator class available on the pyspark.ml.package but guaranteeing class stratification over each fold) (Suen, 2017). The class receives four essential parameters:

- The **numFolds** responsible for the definition of the number of folds to be used by the CV technique. For this work, due to the high volume of information k=3 fold was used.
- The **estimator** parameter takes as input the class object of the classification algorithm, along with its fixed hyperparameters. The following modules were used for the classifiers: RandomForestClassifier, MultilayerPerceptronClassifier, LogisticRegression and LinearSVC.
- The **estimatorParamMaps** is responsible for supporting the hyperparameter tuning of the models through the concept of grid search. The parameter can takes as input another class from the pyspark.ml package named ParamGridBuilder where all the classifier's hyperparameter variations

intended to be validated and assessed are provided. The classifier will be trained and evaluated as many as times as the cartesian product of the hyperparameter provided on the grid.

- The **evaluator** takes as a parameter the class object responsible for the model assessment. For this work the BinaryClassificationEvaluator from the pyspark.ml package was used to assess the performance of each classifier using the area under the curve of the Precision-Recall curves (AUC-PR).

The best set of hyperparameters for a classifier is chosen from the resulting performance assessment against the validation sets from the CV method. However, this result is still an optimistic assessment. Thus, an unbiased performance assessment was performed for the previously chosen most performant hyperparameters but this time over the unseen test set (3).

Using the previously fit object for the training of the classifier, the test set is transformed. The evaluation is carried again using the BinaryClassificationEvaluator class.

4.6.4. Classification Algorithms

With the intent of taking advantage of the distributed processing of the Spark Framework, among the classifiers available in the Spark.ml package the following have been used for several studies and comparisons in the context of intrusion detection by many authors: i) Artificial Neural Networks (Buczak & Guven, 2016; Wang & Jones, 2017); ii) Logistic Regression (Chauhan et al., 2013; Prachi, 2016); and iii) SVM (Buczak & Guven, 2016; Wang & Jones, 2017); iv) Random Forests (Buczak & Guven, 2016; Chauhan et al., 2013; Prachi, 2016; Wang & Jones, 2017).

4.6.4.1. Artificial Neural Networks

The learning algorithm known as Multilayer Perceptron (MLP) is a feedforward Artificial Neural Network (ANN) trained using the backpropagation learning model capable of producing nonlinear decision surfaces (Mitchell, 1997). Among the range of architectures of neural networks, the feedforward neural networks are comprised of multiple layers of neurons, where all the neurons of each layer are connected to all the neurons of the following layer and no connections are established between neurons of the same layers.

Three types of layers are used (Figure 4.10): i) input layer; ii) output layer; and iii) hidden layers. The first one, the **input layer**, is composed of the neurons that accept input values, in other words, the inputs from the features of each instance. The second and third type, the **output layer** and **hidden layers**, are comprised by nodes responsible for producing a linear combinations of their respective input node's weights and bias and applying and activation function to produce an output signal. While the **output layer** is the final layer of the network and returns the result of training or predicting an instance, the **hidden layers,** are optional and located between the input and output layers and allow the model to solve non-linear problems (Buczak & Guven, 2016; Mitchell, 1997; Ussath, Jaeger, Cheng, & Meinel, 2017).

The training algorithm learns the weights and adjusts them using, among others, ruled gradient descent (GD) -based approaches to minimize the error between the produced output and the target output values (Mitchell, 1997). Considering an error surface associated to the hypothesis space of all weight vectors, the algorithm iteratively adjusts the weights for each training instance, by searching for the weight vector that produces the steepest descent along the error surface in an attempt to converge for the global minimum error (Mitchell, 1997). Subsequently, the trained model will be able to predict results over new and unseen instances (Ussath et al., 2017). A more detailed presentation and description of the algorithm is presented in the Appendix 9.9.

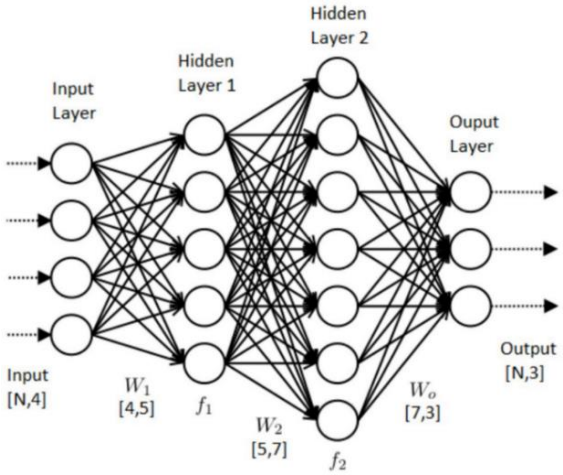


Figure 4.10 - Artificial Neural Network Architecture Example (Bre, Gimenez, & Fachinotti, 2017)

The dataframe API used by the spark.ml package offers a class called MultilayerPerceptronClassifier with an implementation of a Feed-forward Neural Networks algorithm as a MLP classifier using backpropagation as a learning model, a logistic loss function for optimization and two different optimization routines (Apache Software Foundation, 2018g). The algorithm implementation on spark uses as activation functions on the hidden layers nodes the sigmoid (logistic) function, expressed as (eq.3):

$$f(z_i) = \frac{1}{1+e^{-z_i}} \quad \text{(eq.3)}$$

For the output layer, for each of the N nodes, the softmax function is used as (eq.4):

$$f(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{N} e^{-z_k}} \quad \text{(eq.4)}$$

Among the hyperparameters available for the class (Apache Software Foundation, 2018g), the following stand as the most relevant:

- The **layers** parameter defines the number of layers (input, hidden and output) as well as the number of neurons contained in each of them. The number of neurons defined in the input layers must match the number of features used in the training set, and the number neurons used in the output layer must match the number of classes used on the target feature;

- The **solver** defines the optimization routine used for the classifier. Two options are presented, the Minibatch GD method (gd) and the Limited-memory Broyden-Fletcher-Goldfarb-Shanno optimization algorithm (L-BFGS).
- The **stepSize** defines the step to be used for each iteration of optimization (> 0). The stepSize is a scalar value defining the initial step size for GD. All updaters on each iteration use a step size at the t-th step equal to $stepSize/\sqrt{t}$. For this work the default value of 0.03 was used;
- The **maxIter** defines the maximum number of iterations;

For this work, Table 4.15 presents variations of hyperparameters that were combined and assessed in order to find the most performant combinations:

Table 4.15 - Neural Networks Hyper-parameters Tuned

Hyperparameter	Pyspark API	subdivision	Values
Nodes in Hidden Layers	layers	1 hidden layer	[2], [4], [6], [8], [10], [12]
		2 hidden layers [nodes layer 1, nodes layer 2]	[4, 2] [6, 3], [6, 4], [6, 6], [8, 3] [8, 4], [8, 6] [10, 4], [10, 6], [10, 8], [10, 10] [12, 4], [12, 6], [12, 8], [12, 10] [14, 6], [14, 8], [14, 10], [14, 12], [14, 14]
Maximum Iterations	maxIter	-	1000, 10000
Solver Algorithm for Optimization	solver	-	l-bfgs, gd

The training results for all the hyperparameter combinations are detailed in Appendix 9.10. For the most performant combination of hyperparameters during the training stage, several analysis were performed, and the results discussed and compared against the other classifiers of this work.

4.6.4.2. Logistic Regression

The logistic regression algorithm is a widely used learning algorithm, commonly referred as a generalization of the linear regression applied to the binary classification (Gupta & Kulariya, 2016; He et al., 2016; Murphy, 2012).

For a binary classification (binomial family) a logistic function (also known as sigmoid function or log odds) is built from labelled training data, and a probability is estimated over a new unlabeled observation in order to assign it to one of the binary labels (He et al., 2016; Murphy, 2012; Nykodym, Kraljevic, Wang, & Wong, 2019). In other words, given a binary target variable y∈{0;1}, the algorithm models a hypothesis output ($h_\theta(x)$) estimated probability of an observation belonging label 1 (y=1), given the data x and parametrized by θ, as following (A. Ng, 2018; R. Ng, 2018; Nykodym et al., 2019):

$$h_\theta(x) = P(y = 1|x; \theta) = \frac{1}{1+e^{-(\theta^T x)}} \quad \text{(eq.5)}$$

During the training process on each iteration, for training instances $i = 1, 2, \ldots m$, the algorithm will simultaneously update all the weights θ_j values in order to minimize the average cost function (A. Ng, 2018; R. Ng, 2018)

Some implementations, such as Spark.ml package, allow the addition of what is known as a regularization parameter to the cost function expression. The regularization parameters are penalties introduced to reduce the variance of the prediction error in order to avoid overfitting (Nykodym et al., 2019). The implementation of the Spark.ml package allows the inclusion of three different regularization parameters to the cost function through the Elastic Net method. The Elastic Net penalty combines both L1 and L2 penalties and is referenced as beneficial for the overfit control (Nykodym et al., 2019). The L1 penalty, also known as Lasso, penalizes the sum of the absolute values of the coefficients leading to a sparse solution. The L2 penalty, also known as Ridge Regression, penalizes the norm of the model coefficients θ_j, leading to a proportional reduction of the coefficient values simultaneously as the regularization parameter is increased without letting any of the predictors reach zero, while providing more stability and faster computation speed than L1 penalty (Nykodym et al., 2019). Further details over the cost function, and regularization paramters are described on Appendix 9.11.

The dataframe API used by the spark.ml package offers a class called LogisticRegression that supports both binomial and multinomial logistic (softmax) (Apache Software Foundation, 2018g). The class implementation on spark supports both L1 and L2 regularization methods, as well as the elastic net method (Apache Software Foundation, 2018g). Among the hyperparameters available for the class (Apache Software Foundation, 2018g), the following stand as the most relevant:

- The **elasticNetParam** defines the Elastic Net method mixing parameter in a range between 0 and 1. As referred before, when the value is 0 the L2 regularization penalty is applied. For a value of 1 the L1 regularization penalty is applied. Every value between them will mix both penalty methods on the correspondent percentage.
- The **family** parameter defines the label distribution used in the model. Two options are supported: i) binomial; and ii) multinomial. For this work, the binomial family was used since the target is binary.
- The **regParam** parameter defines the value for the regularization method chosen.
- The **maxIter** parameter defines the maximum number of iterations;

For this work, Table 4.16 presents variations of hyperparameters that were combined and assessed in order to find the most performant combinations:

Table 4.16 - Logistic Regression Hyper-parameters Tuned

Hyperparameter	Pyspark API	Values
Regularization Parameter	regParam	0.01, 0.10, 0.50
Elastic Net Penalty Distribution (L1, L2)	elasticNetParam	0.0, 0.25, 0.5, 0.99
Maximum Iterations	maxIter	10, 100

The training results for all the hyperparameter combinations are detailed in Appendix 9.12. For the most performant combination of hyperparameters during the training stage, several analysis were performed, and the results discussed and compared against the other classifiers of this work.

4.6.4.3. Support Vector Machines

The SVM is a popular learning algorithm originally designed for binary classification (Gupta & Kulariya, 2016; K, Aljahdali, & Hussain, 2013; Murphy, 2012) that belong to the generalized family of linear classifiers. The algorithm is based on the concept of finding a maximum-margin separating hyper plane (decision boundary) between the instances of both classes (Buczak & Guven, 2016; Gupta & Kulariya, 2016).

Among the implementations and extensions of the SVM algorithms, the following are used to address binary classification problems (Cambridge University Press, 2008a; He et al., 2016; Murphy, 2012): i) linear scheme SVM; and ii) non-linear scheme SVM. Both are presented with more detailed on Appendix 9.13. For this work, the linear SVM is the most relevant due to the available API used by the spark.ml package.

The linear SVM learning algorithm, derives a discriminant linear function in the feature space from the training instances and their respective classes (Buczak & Guven, 2016; Murphy, 2012). Through the concept of margin, defined by the distance from the decision surface to the closest set of instances, known as support vectors, the learning algorithm is optimized through the maximization of the margin value, as can be seen on the Figure 4.11 (Buczak & Guven, 2016; He et al., 2016). The resulting approach is referred to as hard-margin SVM. This approach might, however, prove to be quite restricting and lead to a less performant generalization capacity of the classifier, especially noted if the data is not linearly separable or noisy (Murphy, 2012). Therefore, an extension of the approach is the introduction of slack variables on the objective function representing the misclassified training instances (Buczak & Guven, 2016; Lardeux et al., 2009; Murphy, 2012). According to Cambridge University Press (2008) and Murphy (2012), the objective function will seek to find the optimal trade-off between the margin width and the number of points required to generate it, through the minimization of the number of the training misclassifications along with maximization of the margin, in what is referred as soft-margin SVM approach.

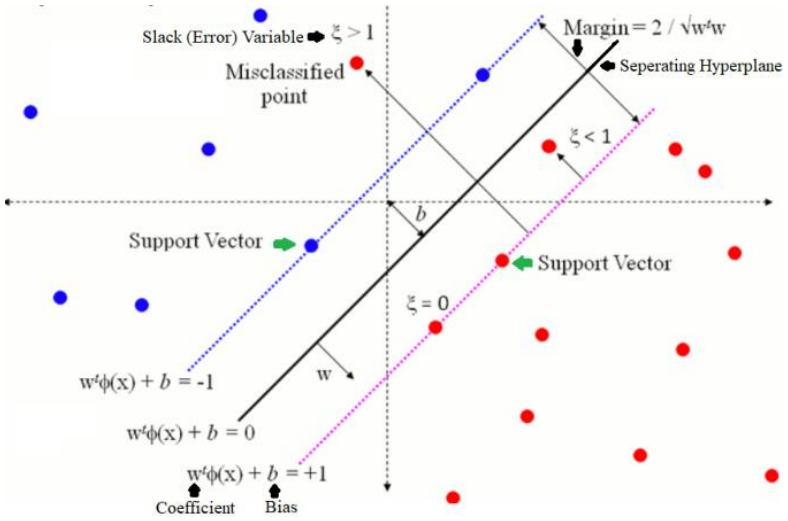


Figure 4.11 - Linear SVM Representation for a Binary Problem (Dey, 2018)

One important distinct property of the SVM classifiers referred by Murphy (2012), is related with the output produced by the algorithm, rather than producing a probabilistic output value (like the other algorithms of this work), the output is a hard-labelling. Approaches such as the one proposed by Platt (2000) are referred to as a way to fit the output into a probability (Murphy, 2012). However, the resulting probabilities are often criticized for producing poorly calibrated results (Murphy, 2012). Thus, in this work, the comparison of SVM classifiers with the remaining probabilistic output-nature classifiers of this work will not use probabilistic-based performance metrics to evaluate the performance among all classifiers.

For this work, it was used the available classification tools provided by the spark.ml package. As such, the package only supports linear SVM through a class called LinearSVC as a binary classifier with L2 regularization method (Apache Spark, n.d.-c; Kulariya et al., 2016). The algorithm optimizes the Hinge Loss function using the Orthant-wise limited-memory quasi-Newton (OWL-QN) optimizer (Apache Spark, n.d.-c). Among the hyperparameters available for the class (Apache Software Foundation, 2018g), the following stand as the most relevant:

- The **regParam** parameter defines the value of the regularization parameter (regParam>0, default 0.01) that allow the trade-off between minimizing the training error and minimizing model complexity (i.e., to avoid overfitting). Making an analogy with the margins concept, the higher the regParam, the smaller the C value and the closer we get from a hard-margin approach. The lower the regParam, the higher the C value and the more relaxed soft-margin approach gets.
- The **maxIter** parameter defines the maximum number of iterations used for the training stage.

For this work, Table 4.17 presents variations of hyperparameters that were combined and assessed in order to find the most performant combinations:

Table 4.17 - Support Vector Machines Hyper-parameters Tuned

Hyperparameter	Pyspark API	Values
Regularization Parameter	regParam	0.01, 0.10, 0.50, 1.0, 2.0
Maximum Iterations	maxIter	10, 20, 30, 50, 100

The training results for all the hyperparameter combinations are detailed in Appendix 9.14. For the most performant combination of hyperparameters during the training stage, several analysis were performed, and the results discussed and compared against the other classifiers of this work.

4.6.4.4. Random Forests

The Random Forests classifier is an ensemble learning method characterized for averaging/voting multiple decision trees estimates, as weak learners, to build a stronger learner (Buczak & Guven, 2016). The algorithm uses a technique called bagging, also known as bootstrap aggregation, to generate multiple and diverse decision trees by randomly choosing with replacement different subsets of data (Breiman, 1996). The Random Forests technique tries to generate as much decorrelated trees as possible by randomly choosing a subset of the available features to split each node (Breiman, 2001). The predictions of each tree are then ensembled through averaging or majority voting, respectively for regressors or classifiers, reducing the variance of the estimations to produce one final output prediction (Gupta & Kulariya, 2016; Murphy, 2012; Timčenko & Gajin, 2017) (Figure 4.12). Details related with the algorithm training process are presented in the pseudo-code in Appendix 9.15.

According to Breiman's work (Breiman, 2001), introducing randomness through bagging and random features can produce significant improvements in classification performance results. The progressive increase of the number of trees used leads to the convergence of the generalization error of the algorithm, while the overall decrease of generalization error of a forest depends on the strength and variance introduced by each individual tree in the forest and low correlation between them.

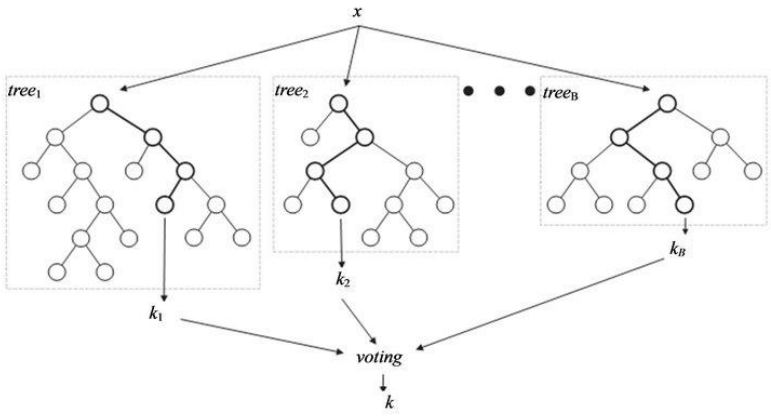


Figure 4.12 - General Architect of Random Forest (Nguyen, Wang, & Nguyen, 2013)

Several authors describe some of the advantages and disadvantages of the algorithm. (Buczak & Guven, 2016; Gupta & Kulariya, 2016) highlights the importance of the algorithm in ensembling and averaging the variance produced by the results of multiple and varied sets of trees generated from different subsets of data and input features, to reduce the risk of overfitting. The higher the number of the tree the lower the variance while bias remains the same. The authors Gupta and Kulariya (2016) refer the importance of some of the advantages inherited from single decision trees algorithms, such as the capability of handling categorical features, binary and multiclass classification, the absence of the need to scale continuous numerical features and its ability to capture non-linear pattern on the data. In the other hand, one of the disadvantages is the loss of interpretability when compared with simple decisions trees (Buczak & Guven, 2016; Prachi, 2016). The algorithm is also quite dependent on the variability methods used to produce decorrelated sets of trees (Buczak & Guven, 2016). Despite its capacity to resist overfitting, Prachi (2016) is cautious when using features with high cardinality, stating that it exposes to the algorithm to the risks of overfitting.

The dataframe API used by the spark.ml package offers a class called RandomForestClassifier with an implementation of the Random Forests algorithm as one of the tree ensemble algorithms available, both as a classifier (binary or multiclass) and as a regressor, using both continuous and categorical features (Apache Software Foundation, 2018g). Among the hyperparameters available for the class (Apache Software Foundation, 2018g), the following stand as the most relevant:

- The **numTrees**, is the number of decision trees to be generated for training and predictions. The higher the number of trees the lower the variance of the predictions at the cost higher training time;
- The **impurityMeasure** is a metric associated with the degree of homogenity of the classes present at a given node. The value is used for the calculation of the information gain of each feature. Two metrics are supported, “gini” and “entropy”. The usage of the gini impurity, provides a less computationally intensive as it does not require to compute logarithmic functions.
- The **maxDepth** represents the maximum depth of any generated tree. According to the Apache Software Foundation (2018), for a value of 0 the tree generated will only have 1 leaf node, for a value of 1 tree will produce ant maximum 1 internal node and 2 leaf nodes. This parameter controls the generation of shallow or deeper trees. The deeper the tree the more fit to the training data it becomes at the cost of increasing training time.
- The **subsamplingRate** defines the percentage of instances sampled with replacement from the training set to generate each learning tree. The values range from 0 to 1. The higher number of instances used to train the trees the higher the training time.

- The **featureSubsetStrategy** is the parameter responsible for controlling the selection of the subset of features used to split each node during the generate each learning tree. For this work the "onethird" and "sqrt" hyperparameters were used.

For this work, table 4.18 presents variations of hyperparameters that were combined and assessed in order to find the most performant combinations:

Table 4.18 - Random Forests Hyper-parameters Tuned

Hyperparameter	Pyspark API	Values
Number of Trees	numTrees	10, 20, 30, 50, 100
Impurity Measure	impurityMeasure	Gini, Entropy
Maximum Depth	maxDepth	4, 6, 8
Sub Sampling Rate	subsamplingRate	0.30, 0.60
Feature Sampling Method	featureSubsetStrategy	onethird, sqrt

The training results for all the hyperparameter combinations are detailed in Appendix 9.16. For the most performant combination of hyperparameters during the training stage, several analysis were performed, and the results discussed and compared against the other classifiers of this work.

4.7. EVALUATION

In this section, for the previously defined models, the classification performance is assessed using a set of statistical metrics throughout all the analysis and experiments conducted on this work. The main objective is to achieve different performance perspectives of the problem, and ultimately, to draw conclusions.

4.7.1. Performance Metrics

According to Buczak and Guven (2016), the following three overall comparison criteria are described as the most commonly used among cybersecurity implementations using Machine Learning methods: i) accuracy and overall classification performance; and ii) training time of the models.

In binary class problems, the **confusion matrix** provides a summary of the correctly and incorrectly classified instances for each class (López, Fernández, García, Palade, & Herrera, 2013). Four statistical metrics are summarized in a 2x2 matrix representation of the classifier's performance over positive and negative instances: i) **True Positives (TP)**: Number of instances with a positive class correctly classified; ii) **True Negative (TN)**: Number of instances with a negative class correctly classified; iii) **False Positive (FP)**: Number of instances with a negative class incorrectly classified as positive class; and iv) **False Negative (FN)**: Number of instances with a positive class incorrectly classified as negative class.

From the confusion matrix, different statistics and insights can be retrieved. According to López and other authors (2013), the **accuracy** rate is one of the most common and practical metrics used for assessment as it provides insights over the percentage of correctly classified instances.

However, for imbalanced datasets problems, one of the main concerns is associated with the evaluation metrics chosen to assess the performance of the classifier due to the different contributions of each class. For some statistical metrics, the results produced from the assessment might be deceptive and lead to the wrong conclusions as the classification of instances associated with the majority will typically produce high-performance statistics that will mask the minority class classification performance. Thus, some metrics might not be the most adequate to assess both classes, as they do not provide the most complete performance information (López et al., 2013). Therefore, the accuracy metric is not the most informative performance statistic for the nature of the problem of this work as it does not distinguish the percentage of the correctly and incorrectly classified instances for the different classes (Davis & Goadrich, 2006; López et al., 2013).

From the confusion matrix, the following statistical metrics were used to assess the performance of the classifiers i) Precision; ii) True Positive Rate (TPR); iii) False Positive Rate (FPR); and iv) F-measure.

- **Precision**: $PPV = \frac{TP}{TP+ FP}$, also known as Positive Predicted Value (PPV), is the percentage of positive predictions instances correctly classified;
- **True Positive Rate**: $TPR = \frac{TP}{TP+FN}$, also known as Recall or Sensitivity, is the percentage of positive instances correctly classified;
- **False Positive Rate**: $FPR = \frac{FP}{FP+TN}$, is the percentage of negative instances incorrectly classified;
- **F-measure**: $F_\beta = (1 + \beta^2)\frac{PPV \times TPR}{\beta^2 \times PPV + TPR}$, combines the precision and TPR in a single metric. According to López and other authors (2013), a popular choice is to use β=1, the measure is known as F1-measure combines precision and recall with equal contribution. However, if more weight is desire over the precision metric β=2 can be used.

When evaluating binary problems another widely known and important approach that produces a unified measure from some of the previously presented measures is the **Receiver Operating Characteristic (ROC)** plot (Davis & Goadrich, 2006; López et al., 2013). This approach combines the TPR and FPR metric for every threshold in a two-dimensional graphical representation, as shown in Figure 4.13. This approach evidences the trade-off between choosing different thresholds to assess the TPR and FPR for a classifier (Cieslak et al., 2006; López et al., 2013). An important performance metric that can be extracted from this representation is the area under the ROC curve (AUC-ROC) (eq.6), as it provides a single metric for the classifiers average performance across all the thresholds, allowing a direct performance comparison between different classifiers (Chawla, Bowyer, Hall, & Kegelmeyer, 2002; Cieslak et al., 2006). The closer the curve is to the upper-left corner, the higher the area under the curve (AUC) value and consequently the higher the classifier's performance when compared to others (Davis & Goadrich, 2006). According to Chawla et al.

(2002), the usage of ROC curves or other similar techniques is the most suitable method to assess the performance of a learning algorithm when facing imbalanced datasets with unequal error costs.

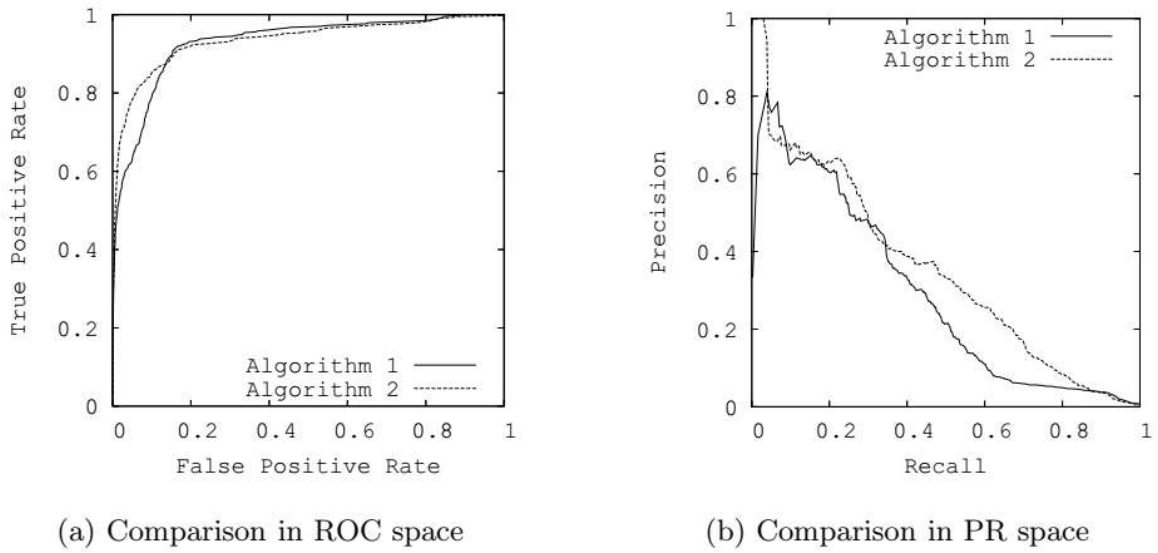
$$AUC - ROC = \int_0^1 \frac{TP}{TP+FN} d\left(\frac{FP}{FP+TN}\right) = \int \left(TPR \cdot d(FPR)\right) \quad \text{(eq.6)}$$

$$AUC - PR = \int_0^1 \frac{TP}{TP+FP} d\left(\frac{TP}{TP+FN}\right) = \int \left(PPV \cdot d(TPR)\right) \quad \text{(eq.7)}$$

However, despite the ROC curve's wide usage as a classifier's performance metric, some authors argue that for imbalanced datasets the most suited graphical method is the **Precision-Recall (PR) curve** graphs (Davis & Goadrich, 2006; López et al., 2013). The work developed by Davis and Goadrich (2006) compares the relationship between the Precision-Recall and ROC curves and argues the later ones can be overly optimistic when assessing the performance of classifiers over datasets with overly imbalanced class distributions. The PR curves have been cited as an alternative to ROC curves for classification problems where the class distribution is highly skewed (Davis & Goadrich, 2006). For these scenarios, the PR curves can be more expressive of the performance representation and unveil differences between classifiers unnoticed on the ROC curves (Davis & Goadrich, 2006). Similar to the ROC curve approach, the PR curves combine precision and TPR metrics in a graphical representation for every threshold in a two-dimensional graphical representation, as shown in Figure 4.13 A single average performance metric, the area under the PR curve (AUC-PR) (eq.7), can be computed in order to assess and compare different classifier's performances.

Comparing both ROC and PR curves, according to Davis and Goadrich (2006), it is possible to define a dependency between the ROC space and the PR space. The author concludes that for a given learning algorithm both ROC and PR curves possess the same points. Given this information, the author progresses to conclude that if a PR curve establishes a relationship of supremacy over another, then the same dominance is present on the ROC curves, while the opposite might not be true (Davis & Goadrich, 2006). Another important conclusion of the same authors is that the linear interpolation between points of the same curve is not possible in the PR curves, as opposed to the ROC curves, due to the non-linear relationship between them (Davis & Goadrich, 2006). Lastly, the same authors state that the optimization of AUC-ROC is not guaranteed to optimize the PR curve (Davis & Goadrich, 2006).

Another important indicator is the processing **time consumed for training** each classifier. Time is a valuable resource and as such it has to be measured and compared among classifiers as a metric of performance. The time consumed by each classifier over their capacity to correctly classify each instance is a trade-off that has to be managed in a production environment. Even though it is not the main objective of this work to find a classifier capable of fitting a certain range of time each time it has to be trained, it certainly provides important insights among the most performant models (Buczak & Guven, 2016).



(a) Comparison in ROC space (b) Comparison in PR space

Figure 4.13 - The Difference Between Algorithms Using ROC and PR Space (Davis & Goadrich, 2006)

For this work, the following points were considered:

- Having the dataset class distribution highly imbalanced, the classifiers performance comparison will use the PR curves approach over the ROC curves as a primary performance metric to assess the models.
- The models capable of producing a probabilistic output, the Random Forests, Neural Networks and Logistic Regression models, were trained and had the hyperparameters tuned to maximize the average CV assessment using the AUC-PR.
- For the SVM model, where the output is not probabilistic, the training and hyperparameters were tuned to maximize the f1-measure.
- For the comparison between all classifiers, for each of the probabilistic output classifiers, the threshold that maximized the f1-measure was chosen.
- The f1-measure was chosen over the f2-measure due to the importance given by the SOC team towards the misclassified requests. As both present equal importance for the problem, with no detriment of one to the other, the assessment false positives should not weight more than missing the correct classification of a real server threat request, given by the false negatives.
- All the remaining metrics, precision, TPR, FPR and f2-measure were used to support the analysis of the previous primary metrics.

4.7.2. Performance Analysis

Three analysis were conducted in order to breakdown the classifiers performances over different perspectives used for the problem: i) Performance by classification algorithm; ii) Performance by feature selection; iii) Performance by processing time and storage format.

The first analysis, **Performance by classification algorithm**, intends to identify the best performing classifier over the 4 different algorithms used from the Spark ML library for our classification problem: Artificial Neural Networks, Logistic Regression, SVM and Random Forests.

The second analysis, **Performance by feature selection**, intends to study the effects of performing feature selection over the most performant classifier in order analyze the features that most contribute for the classification framed by our business context.

The third analysis, **Performance by processing time and storage format**, intends to extract insights over the time taken for each learning algorithm to be trained versus the gains in term of classification performance. Starting with the comparison of the different data formats and respective gains in storage, the loading of the data to Spark DataFrames and its whole preprocessing is timed and compared (using the most performant cluster and spark-submit parameter configuration). Next, the different storage formats are assessed in their elapsed time over the training of the four different classifiers used (again, using the most performant cluster and spark-submit parameter configuration). Finally, for the most performant file format and classifier in the whole pipeline, in term of elapsed training time, different configurations of the cluster and spark-submit parameters are compared, in order to better compare and understand the different configuration decisions used.

5. RESULTS AND DISCUSSION

The results and discussion chapter is the culmination of this work's development to answer the challenge of supporting the organization's SOC team on the task of identifying server threat requests of a particular service of the financial organization, at the SIEM level of infrastructure. The chapter comprises a brief summary of the training results that lead to the choice of the most performant hyperparameters and three different analysis over the classifiers used: i) Performance by classification algorithm; ii) Performance by feature selection; and iii) Performance by processing time.

As a summary, the following methods, approaches and technologies were used for all the analysis performed:

- The large volume of data involved was handled using the **Spark Framework** for a distributed processing system, with **YARN** as RM, and the original CSV files store throughout the DataNodes in **HDFS**;
- Besides the original CSV data type collected, the whole framework was developed and evaluated using two other different file formats generated from the original, the **ORC** (**Native**, **ZLIB** and **SNAPPY**) and **Parquet** (**Native**, **GZIP** and **SNAPPY**).
- The original dataset was loaded into Spark DataFrames and preprocessed in order guarantee the data quality and consistency, by handling the missing values, inadequate, uninformative and redundant data, and to significantly reduce the high cardinality present in all the categorical features through categorical binning (bucketing).
- Two methods were used to assess the feature importance and selection, the **Pearson's Chi-Square Test** as filter approach before the classifier training, and the **Random Forests** classifier itself as a wrapper approach. For the first one, the transformed dataset was tested for independence in order to perform feature selection by filtering out irrelevant features for the prediction of the target variable. However, no binned features were removed due to their rejection of the null hypothesis. For the second method, the results are produced from training a classifier, and therefore, will be under analysis in this chapter.
- The dataset was split into **training set (70%)** and **test set (30%)**, and during the training stage of the classifiers, the hyperparameter tuning and training overfit control used the **3-Fold Cross Validation**;
- For the training of the classifiers, the imbalanced dataset problem was handled using the **BEV** to produce 105 balanced training subsets and the results ensembled. The test set was kept imbalanced;
- Four classifiers were used: **Artificial Neural Networks (ANN)**, **Logistic Regression (LR)**, **SVM** (with linear schema), and **Random Forests (RF)**;
- The primary evaluation metrics used to assess the scoring performance of the classifiers, capable of producing probabilistic outputs, was the **area under the curve of precision-recall curve** (AUC-PR). The only exception stands for the SVM were the **f1-measure** was used, due to the non-probabilistic nature

of the output produced by the classifier. For the comparison between all classifiers, for each of the probabilistic output classifiers, the threshold that maximized the f1-measure was chosen.

For the classifiers and methods previously mentioned, the following Table 5.1 lists the average most performant set of hyperparameters achieved during the training of the classifiers for the respective CV applied. All the training results are detailed in the Appendixes 9.10, 9.12, 9.14 and 9.16:

Table 5.1 - Best Set of Hyper-parameters Tuned for the Validation Set for Each Classifier

Classifier	Hyperparameters	CV Evaluation Criteria
Artificial Neural Networks (ANN)	layers: 2 layers [8, 6], maxIter=10000, solver =l-bfgs	Max AUC-PR
Logistic regression (LR)	regParam=0.10, maxIter=100, elasticNetParam=0.25	Max AUC-PR
Support Vector Machines (SVM)	regParam=0.01, maxIter=20	Max f1-Measure
Random Forests (RF)	impurityMeasure=gini, featureSubsetStrategy = sqrt, subsamplingRate=0.3, maxDepth=8, numTrees =100	Max AUC-PR

All the analysis performed in this chapter are developed using the most performant set of hyperparameters for the respective classifiers.

5.1. Performance by Classification Algorithm

The first analysis, the **Performance by classification algorithm**, is focused on answering the primary and most important part of the challenge of this work, the development of an automatic and efficient solution for the identification of server threat requests over the SIEM logs. Table 5.2 summarize the scoring performances achieved for the classifiers capable of producing probabilistic outputs and Table 5.3 summarizes the scoring performances achieved for the non/probabilistic, both over the test set.

Table 5.2 - Scoring Performance Over the Test Set for the ANN, LR and RF

Classifier	3-Fold CV Criteria	AUC-PR	AUC-ROC	Threshold	F1	F2	TPR	FPR	PPV
ANN	Max AUC-PR	0.99765	0.99997	MaxF1(80%)	0.95658	0.98216	**0.99999**	0.00086	0.91678
LR	Max AUC-PR	0.99502	0.99995	MaxF1(30%)	0.85643	0.93705	0.99979	0.00317	0.74903
RF	Max AUC-PR	**0.99935**	**0.99999**	MaxF1(70%)	**0.99278**	**0.99695**	0.99975	**0.00014**	**0.98592**

Table 5.3 - Scoring Performance Over the Test Set for the SVM

Classifier	3-Fold CV Criteria	F1	F2	TPR	FPR	PPV
SVM	Max F1	0.95713	0.98130	0.99810	0.00083	0.91940

Analyzing the previous Tables 5.2, for the classifiers capable of producing **probabilistic outputs**:

- The overall average scoring performance from the 105 models trained, of the three classifiers, in terms of AUC-PR presented high values above 99%.
- The classifier with the best scoring performance was the **Random Forests (RF)** with an average overall AUC-PR value of 99.935%. The RF model presents a small scoring margin advantage over the remaining classifiers with scoring values 0.170% higher than the ANN model (99.765%) and 0.435% higher than the LR model (99.502%).
- An interesting observation can be extracted regarding the AUC-ROC. For this metric, the same order of performance dominance as the AUC-PR, between classifiers is verified with 99.999%, 99.997% and 99.995%, for RF, ANN and LR, respectively. Thus not contradict the conclusions drawn by Davis and Goadrich (2006) regarding the dominance of a classifier over another through the AUC-PR values, and the subsequent validation of the same dominance through the AUC-ROC values.

$$AUC\ PR_{RF} > AUC\ PR_{ANN} > AUC\ PR_{LR} \xrightarrow{Therefore} AUC\ ROC_{RF} > AUC\ ROC_{ANN} > AUC\ ROC_{LR}$$

Analyzing the previous Tables 5.2 and 5.3, the classifiers capable of producing **probabilistic and non-probabilistic outputs are compared** based primarily on the f1-measure, the metric used to choose the average most performant SVM model over the CV. For the probabilistic output classifiers, the thresholds were chosen for the comparison maximize the same metric over the CV. The following observations are withdrawn from the comparison:

- The overall average scoring performance of the four classifiers in terms of f1-measure presented values above 85%.
- Analyzing the classifiers performances over the f1-measure, the initial conclusion regarding the average most performant classifier stands in favour of the **Random Forests** with an average overall value of 99.278%. The classifier produced for more than 18.6 million instances that compose the test set, the highest number of correctly classified instances for both classes only misclassifying 2522 normal service requests, as false positives and 45 server threat requests, as false negatives. Therefore, resulting in an average overall highest values for the f1 measure, f2 measure (99.695%), precision (98.592%), and the lowest FPR (0.014%).
- For the f1-measure, the RF model presented a scoring margin advantage over the second most performant classifier with scoring values 3.73% higher than the SVM model with 95.71%, an 3.78% higher than the ANN model with 95.66%. For all the remaining assessment scoring metrics, both SVM and ANN models present performance differences similar to the f1-measure, with less than 1% advantage for the SVM over the PPV, f2-measure and FPR.
- The SVM model presented the highest TPR performance of all four classifiers due to a small number of misclassified server threat requests, less than 35 false negatives when compared with the RF model.

However, this metric alone does not account for the number of false positives where the RF model outperforms, where the SVM misclassified more than 15000 instances as false positives.

- From the four classifiers, the Logistic Regression model presented the lowest scoring performance in any of the used metrics for scoring assessment.

Making a **comparison with analogous studies**, the only work with a similar approach to the intrusion detection problem using the SIEM infrastructure, to the extent of this work's author knowledge, are the studies developed by Suh-Lee et al. (2016). Using Machine Learning algorithms, the authors performed classification tasks to identify different intrusions attempts over the artificially simulated network environment dataset from the Packet Clearing House known as SKAION 2006 IARPA Dataset. From the conclusions elaborated for the study, one that most interest brings for this work is the algorithms that achieved the best performances in terms of precision, recall, specificity and accuracy. Sharing many of the same features, for the scenario where all the information extracted by the author was used for classification, the top three most performant algorithms were tree-based algorithms with the Random Forest classifier topping the list, as in this work, outperforming other classifiers including perceptron based. The authors argue that, for their dataset, the classifiers performance is significantly affected by the type of data used, and therefore for the data that contains categorical features or is a mixture of features, Random Forests displayed the best performance, as it is verified in this work.

5.2. Performance by Feature Selection

The second analysis, the **Performance by feature selection**, is focused on the most performant classification algorithm, the Random Forests, and the feature importance produced by the classifier to be used as feature selection wrapper method. Although the number of features might seem too low to justify a feature selection analysis (only six), the objective is to understand the influence of the different contributors for the problem and extract insights over their role in the overall business problem, the prediction of intrusion attempts.

Through an iterative process, the training of the Random Forests classifier was used to produce a ranked output of the contribution of each feature for the scoring performance. Starting with all the features, each following iteration the least contributing feature from the previous iteration was removed and assessed the classifier performance against the test set for the AUC-PR metric.

The following Figure 5.1 represents the summarization the scoring performances achieved over the test set for each iteration:

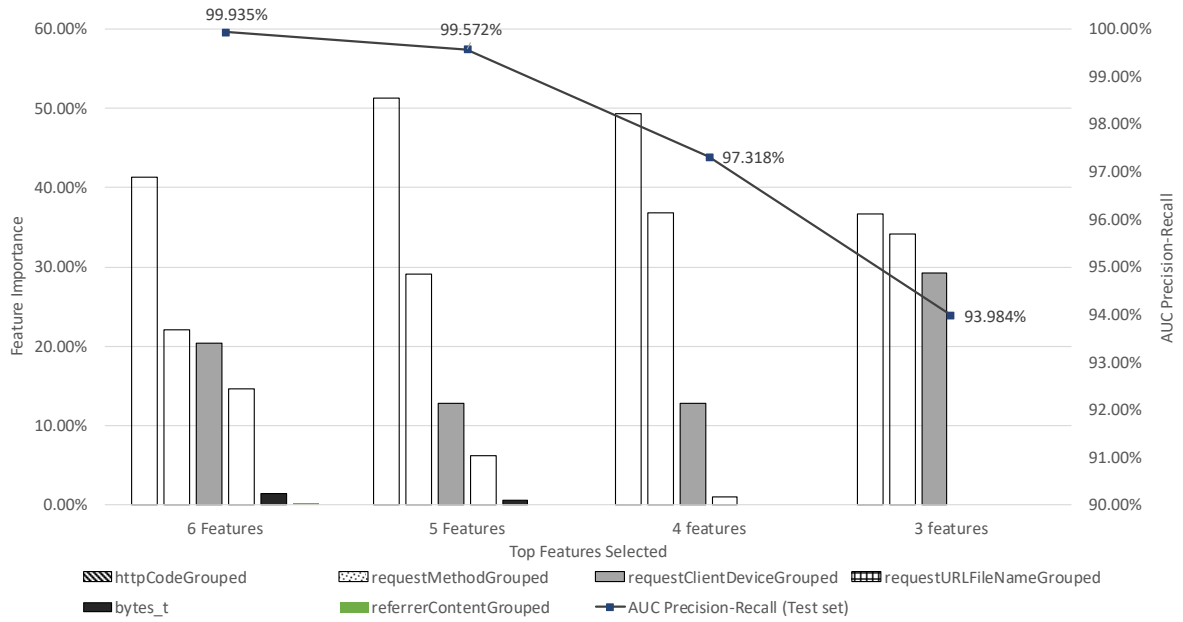


Figure 5.1 - Features Importance Performance Analysis for the Test Set using Random Forests

Analyzing the previous Figure 5.1 it is possible to extract several conclusions related to the results:

- The first and most notorious observation is that using all the features have a positive impact on the learning algorithm's performance against the test set. A steady increase from 93.984% to 99.935% of the AUC-PR values is noted as the number of features increase from three to six, according to its order of importance, leading to the overall best performance scenario of the learning algorithm using all the six features.
- The second observation is that the feature "httpCodeGrouped", representing the HTTP codes grouped, is the most relevant of the six for the learning algorithm's performance. In every scenario tested with the number of features, this feature played an influence for the classifier's performance between 36% to 51%. Taking in consideration the feature meaning for the reality of the problem of this work, this result was to be expected as the type of server threat requests associated, vulnerability search, are heavily based on trial and error attempts, where in each attempt the perpetrator tries to extract insights over the server response to further guide the next attempts. Therefore, a high number of client errors server request response 4xx are generated, contrasting with the high volume of non-threat successful server request responses of the type 2xx.
- The third observation is related to the features that presented the least contributions for the classifier's performance, the "referrerContentGrouped" and "bytes_t". Each contributed less than 2% for the overall performance. However, the small contribution of these two features allowed the classifiers' performance to increase from 97.32% to 99.93% in terms of AUC-PR. The perception of these two features over the reality of our business problem and the results obtained are not so trivial as the previous observation. For the threat server requests, the high number of HTTP 4xx responses

present is in general represented by a low range of bytes response, since the request was not granted, while a granted request will respond with the size of the content requested. As for the referrer content, the values that contain a message are all associated with non-mobile requests, narrowing the range of potential perpetrators.

- The last observation is related to the overall feature importance order and contributions on each scenario. Along the iterations, as the number of features is decremented, the ranking order of the feature contribution does not change. However, the relative contribution of each feature for the evaluation metric changes with more pronunciation on the three features test, when compared with the remaining. While the contribution of each feature on the scenarios of six to four features has an average value of 16% to 25% with standard deviations between 13% to 19%, the three-feature scenario with the worst performance scores have an average feature contribution of 33% with a standard deviation of only 3%.

Overall, it can be concluded that training the classifier with all the six features produces a positive influence for the classifier to achieve the most performant scoring values against the test set.

5.3. Performance by Processing Time and Storage Format

The third analysis, the **Performance by processing time and storage format**, is focused on the Big Data problem itself, with the storage the comparison of different storage formats and their respective elapsed time for each of the stages of the whole pipeline, with special emphasis on the training time of the models, and over different node configurations and Spark-submit parameters

Starting from the originally collected CSV files from the SIEM, three data types where tested seeking to evaluate the performances of the different stages of the pipeline over row-oriented data (CSV), column-oriented data (ORC and Parquet) and with different compressions applied (Native, ZLIB and SNAPPY for ORC, and Native, GZIP and SNAPPY for Parquet).

The following Figure 5.2 summarizes the storage gains achieved for each of the file formats used in comparison with the default format, the CSV, and the elapsed time performances achieved during the stages prior to the training of the learning algorithms, the **loading of the data** to Spark DataFrames and its whole **preprocessing**, using the most performant cluster configuration and spark parameters (the 3 worker nodes available and a balanced set of parameters as Spark-Submission as it will be analyzed on the last part of this subchapter):

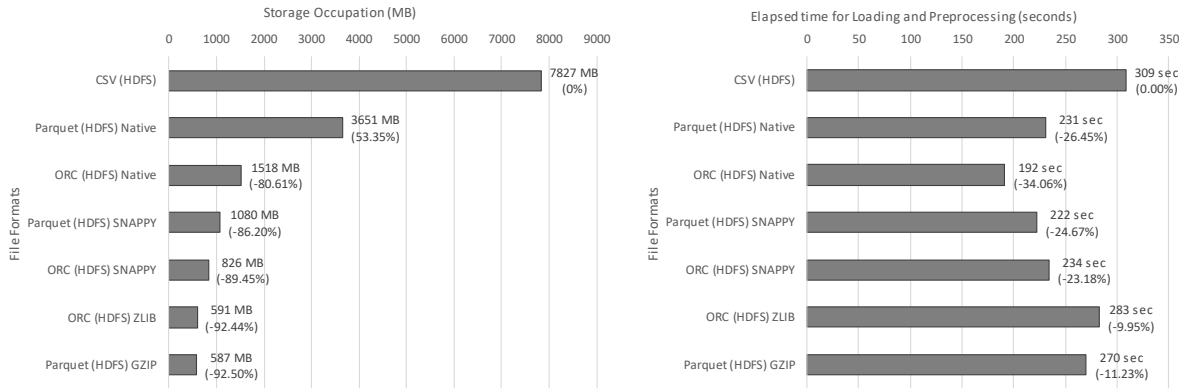


Figure 5.2 - File Format Storage Gains vs CSV (a), and Elapsed Times for Loading and Preprocessing (b)

Analyzing the previous Figure 5.2 it is possible to extract several conclusions related to the results:

- Overall, on **average**, changing the data orientation from rows (CSV files) to columns (ORC and Parquet tables) allowed data storage reductions of 82.43%, from 7.837GB to 1.375GB.
- The compression applied to the ORC and Parquet tables produced the most relevant storage reductions. In comparison with the original CSV files of 7.827GB (without accounting for the HDFS replicas), the **ORC (ZLIB)** and **Parquet (GZIP)** table compressions resulted in reductions to 591MB and 587MB, respectively, allowing storage savings between 92.44% to 92.50%.
- As for the elapsed time for the data loading and preprocessing prior to the training, using column-oriented format data, allowed elapsed time performance reductions between 11% to 34% when compared with the CSV. The file format that was most performant was the **ORC Native** table (with no compression), consuming in average 206 seconds (less than 4 minutes), less 34.06% than the worst file format, the CSV file with 312 seconds (more than 5 minutes).

Following the data loading and preprocessing, the next Figure 5.3 represents the summarization the average elapsed time for the training of each of the learning algorithms, for the previously mentioned set of hyperparameters, while making a comparison against their respective scoring performance over the test set, using the most performant cluster configuration and spark parameters (the 3 worker nodes available and a balanced set of parameters as Spark-Submission as it will be analyzed on the last part of this subchapter):

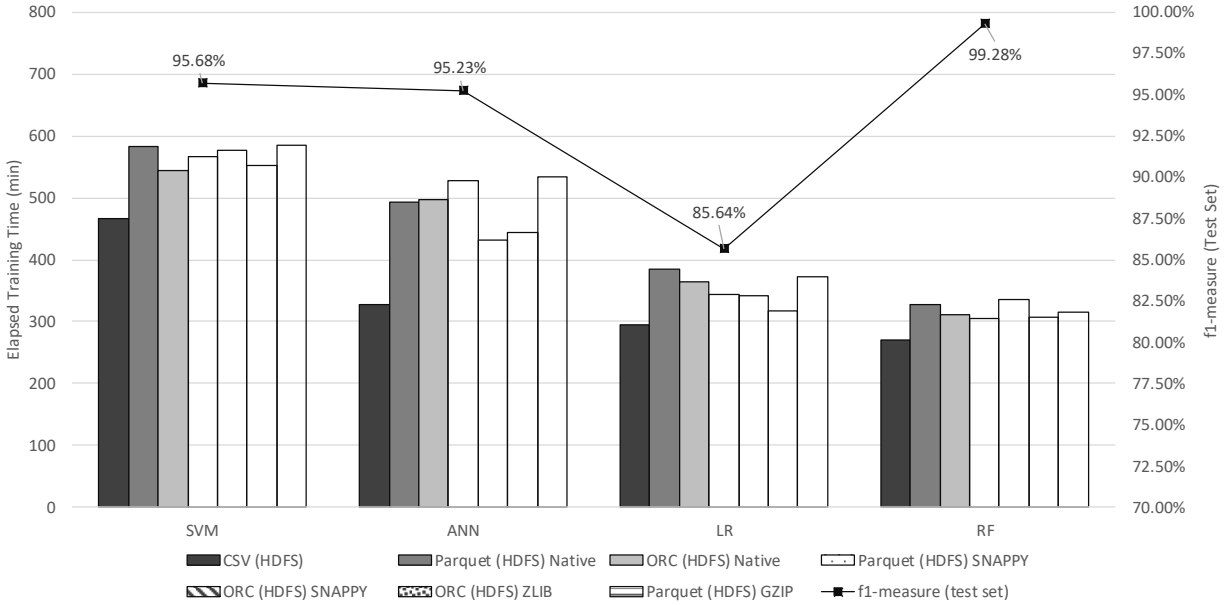


Figure 5.3 - Elapsed Training Time for All the Classifiers and All the File Formats vs f1-measure (test set)

Analyzing the previous Figure 5.3, the **processing times of the classification algorithms** implemented in the Spark.ml package for the dataset used in this work and for the experimental setup previously mentioned lead to the following conclusions:

- Overall, and using the average value for all the used storage formats, the first conclusion that can be withdrawn is related with the training elapsed time for the classification algorithms that produced the best and worst performant results. The average processing times for the **RF and LR models** achieved the fastest values, respectively of 310 and 345 minutes, contrasting with the ANN and SVM models which elapsed the most times for the training stage, with average values between 26% to 44% worse than the LR and RF models, respectively 465 and 553 minutes.
- Crossing the information previously gathered related with test set performance of the four classifiers, the **RF model** presented the average best scoring evaluations while requiring the least training times.
- For the remaining classifiers, the ANN, SVM and LR models, the choice is a tradeoff between scoring performance and elapsed training time performance. The SVM model while providing the second average best scoring performance against the test set, it consumed the largest average amount of time to be trained. When compared with the ANN model, which produced similar scoring results, the average elapsed time for training is 16% worse. The LR model presented the lowest of the classification scoring evaluations but outperformed both ANN and SVM models on the training elapsed time.
- As for the different file formats used to load the DataFrames, the results collected display the most performant training times for **CSV file** format in every one of the four evaluated algorithms, requiring on average between 16% and 25% less time than all remaining data formats.

- From a complete framework pipeline perspective, the elapsed training time compared with the loading and preprocessing times in the training best-case scenario (CSV format with Random forests), represented more than 98% of the total processing time of the framework (52 times than the loading and preprocessing stages together).

As a final note of this analysis, aside from the undeniable choice of the random forests as the most performant classifier in terms of scoring and training elapsed times, it is important to comment the tradeoff between the storage gains and the processing elapsed times. Depending on the future constrains of the technological environment, enhanced performances can be achieved. If in one hand, the available storage is the main constraint, the usage of ORC (ZLIB) and Parquet (GZIP) tables is the data format to be chosen with significant impact over the storage occupied. In the other hand, being the training of the classifiers the biggest bottleneck in terms of elapsed time of the whole pipeline, if the time window between retraining of the chosen classifier is the main constraint, the CSV files provide the most performant choice. No prediction times after training were considered on this analysis due to its speed of processing, almost instantaneous even over the test set with almost 19 Million records.

The last results analyzed are referred to the **cluster distributed processing** of the cluster itself. For the most time-consuming step of the pipeline, the elapsed times for training the classification algorithms, and for the most performant file format and classifier, the CSV and Random Forests, the elapsed times are compared over the usage of different node configurations and Spark-submit parameters.

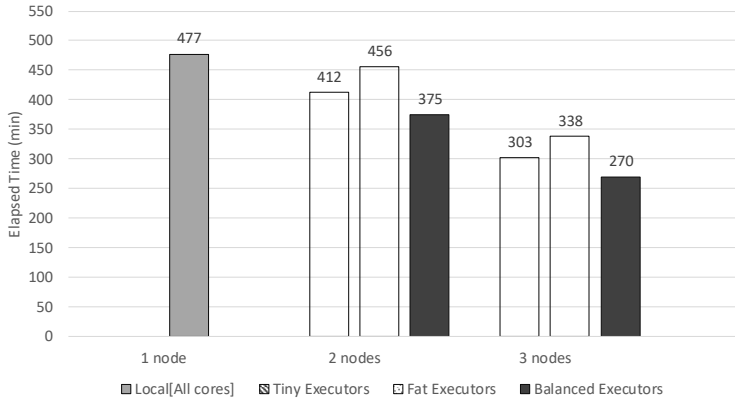


Figure 5.4 - Elapsed Training Time for CSV and RF for Different Node and Spark Configurations

Analyzing the previous Figure 5.4 it is possible to extract the following conclusions related to the results:

- Overall, the first conclusion that can be withdrawn is that using distributed processing is beneficial for the framework's training elapsed times. Using the two and three worker nodes available displayed better performances than using only one isolated node. On average and comparing with running the training on a single node local mode, using two worker nodes reduced the elapsed times by 13% (from 477 to 414 minutes), while using three worker nodes reduced 36% (from 477 to 303 minutes).

- The best performance was achieved using the **3 worker nodes available** (maximum), with a **balanced** set of parameters as Spark-Submission, which has into account the estimated overheads of the OS, YARN and AM. When compared with the baseline, the single node configuration, the training processing times reduced by 44%, from 477 to 270 minutes.
- Between using tiny executors, fat executors or a balanced executor's configuration, the balanced combination outperformed the previous, with less elapsed times between 9% and 20%.

One final comment over this last analysis is that, even though the single node elapsed times was outperformed by the distributed processing experiments, it performed quite well when compared with the resources that were expended. There is a resource penalty, commonly referred to as an overhead, for the use of distributed processing such as the one used in this work, that does not affect the processing when running in local mode. When distributing the processing, the resources consumed by the YARN management and the launched AM, are quite noticeable for a cluster configuration with a low number of cores, such as the one used in this work. From the available resources for processing, the processing gains from using three worker nodes instead of only one (not distributed), only produced a training time reduction less than 50%, most likely due to the distributed processing overheads. This penalty is not completely fixed but is configurable and does not change a lot. Therefore, drastic performance increases are expected with more worker nodes, but more importantly more cores per worker node due to the overheads.

Wrapping the main conclusions of this third analysis, the highest storage gains were from the column-oriented formats ORC with ZLIB compression and Parquet with GZIP compression. The most performant file format during the data loading and preprocessing of the data prior to the classifier training was the ORC with no compression. For the training of the classifier, the most time-consuming stage of the pipeline, the CSV file format and Random Forests classifier outperformed the remaining combinations. As for the cluster configuration and spark-submit parameters, the most performant choice was using the three available worker nodes distributed processing with a balanced parameter configuration of executors and cores.

6. CONCLUSIONS

In this chapter, a summary of the most relevant challenges, decisions, and insights extracted from the analysis performed with the developed framework solution are presented.

During the last years, the world has witnessed an increasing growth in the number of reported cyber attacks along with their estimated sustained damages. These nefarious activities reached a point where organizations and countries can no longer be passive about it and started to heavily invest in cybersecurity campaigns, expertise teams, products and services. Among the most influential and sensitive business areas affected by this threat is the financial sector, where the financial organization of this dissertation work is framed. To solve this problem, several technological solutions have been developed and studied, capable of acting in many different layers of a system, mainly through IDS focused on the identification of threat attempts.

The development of this dissertation work focused on a financial service of an organization that operates on the international markets in the payment systems industry, that allows end users and merchants to access a payment service through mobile or POS devices. For this work, a predictive framework solution was developed capable of performing intrusion detection tasks (classification) over the exponential growing data log events collected by the SIEM through a Big Data processing infrastructure, from the Apache Web Servers for the financial service. As such, it is possible to state that the proposed objectives have been successfully achieved. Through the challenges overcome during the development of the framework solution, technical competencies have been acquired in the various dimensions that comprised the complexity of this work, with special emphasis on the implementation of a complete Data Mining pipeline over a Big Data structure framework.

Through this work, a distributed processing solution over a four-node cluster using, among other tools and services, the Apache Spark as the processing engine along with the ML package to perform classification tasks, was developed. The data was collected in CSV log files from SIEM and was stored in the cluster in HDFS. Several performance studies were conducted using not only the original CSV file format in HDFS, a row-oriented format, but also from Hive tables populated with the CSV data and reorganized into a columnar-oriented format using ORC and Parquet with different types of compressions associated (ZLIB, SNAPPY and Native for ORC, and GZIP, SNAPPY and Native for Parquet). The results showed a significant storage size reduction using a columnar-oriented format in comparison with the CSV, with the most significant values achieved for the ORC table with ZLIB compression and Parquet table with GZIP compression, allowing storage savings between 92.44% to 92.50% (from 7.827GB to 591MB and 587MB).

The understanding of the collected data provided the first great technical interaction with the SOC team through the interpretation and comprehension of the data and their role and significance in the complex network and security environment. From the data exploration the most relevant finding, yet expected for problems of this nature, was related to the binary target variable imbalanced contribution, with the number of threat server requests contributing less than 1% for the whole dataset, which triggered a detailed study of several methodologies to handle the data imbalance and the underperformance of training algorithms when exposed to a biased training.

The next stage, the data preparation was a complex and iterative process with a significant collaboration of the SOC team's experience in the data cleaning, validation and transformation. One of the major challenges of this work was related to the high number of categorical features and their high cardinality. A substantial reduction of their cardinality was performed through a fine-tuned categorical binning in order to significantly reduce the number of levels while retaining the discriminatory power of each feature. In the most noticeable display of the binning performed, the feature "requestUrlFileName", associated with URL server requests, was transformed from having almost 7000 levels to only 4. As for the feature selection method, the chi-square test of independence was applied prior to the model training as a filter method. The tests results indicated that all binned and transformed categorical features rejected the null hypothesis and, therefore, were potentially relevant for predicting the target variable and none was filtered out.

During the modelling and evaluation stage, the data was split into training and test set while keeping its stratification. The test set was kept imbalanced, but the training set was handled differently with the implementation of the methodology of the BEV to produce ensembled results from 105 balanced datasets without generation of synthetic data and covering all the available data. For the modelling of the learning algorithms, four different classification algorithms were trained (ANN, LR, SVM and RF), and the best set of hyper-parameters of each, chosen from the 3-fold CV, using the AUC-PR for the ANN, LR and RF and the f1-measure for the SVM due to their inability of producing probabilistic outputs. From this point, the first analysis of the final results was produced where the different classifiers were compared over their scoring performance, with their validation threshold chosen to maximize the respective f1-curves (ANN, LR and RF). The final results show that the fittest classifier for the problem, dataset and metrics used was the Random Forests, with an overall f1-measure over the test set of 99.278%. From the 18.6 million instances that compose the test set, only 2522 normal service requests were misclassified, as false positives, and 45 server threat requests, as false negatives.

The second analysis produced focused on the performing and iterative feature selection from the produced feature importance of the random forests, the classifier with the best scoring performance. Through the analysis of the feature contribution for the scoring results, an iterative process of removing them one by

one was performed in order to extract the conclusion of their role in the classification. The main results achieved allowed the conclusion that the three most contributing features, related with the HTTP code response, the request method used, and the device used by the client to make the request, were alone responsible for 93.98% of the AUC-PR scores achieved for the test set. However, the remaining features used for the training had their contribution justified on the performance of the algorithm, allowing the classifier to ascend their AUC-PR performance from 93.984% to 99.935%.

The last analysis produced focused on the performance by the processing time of the whole framework. The elapsed times over the different stages of the whole pipeline of the framework were registered and compared against each of the file formats initially used by the framework. The objective was to understand if the gains in storage would also provide benefits for the spark processing times as well as to analyze the different cluster and spark configurations used. The results produced showed that the highest storage gains were from the column-oriented formats ORC with ZLIB compression and Parquet with GZIP compression above 92% when compared with the CSV (HDFS) format. As for the loading of the data into DataFrames and all the preprocessing tasks performed prior to the training of the algorithms, overall using any of the column-oriented formats produced elapsed time reductions of at least 11%, with the best performance to be achieved for the ORC Native table (with no compression), consuming in average 206 seconds, less 34.06% than the worst file format, the CSV file with 312 seconds. The elapsed time for the training of the classification algorithms represented above 98% of the whole processing time of the framework. For the classification, the row-oriented CSV file format outperformed any of the column-oriented formats in terms of elapsed time with 339 minutes, an average between 16% to 25% less. As for the classification algorithms, overall the Logistic Regression and the Random Forests consumed the least time to train, with an average of 345 and 310 minutes, while having the worst and the best scoring performances against the test set associated, respectively. From the cluster configurations and spark-submit parameters tested, the most performant choice was using the three available worker nodes distributed processing with a balanced parameter configuration of executors and cores.

The main contributions of this dissertation work for the cybersecurity field consisted on the implementation of a system for threat detection at the SIEM level, a centralized log system typically located at the end of the data pipeline that reaches the SOC team, contrasting with the typical implementation of an IDS as the first line of defense of the whole system. The emphasis is not the traditional preventive action against incoming server requests and the analysis at the packet level, but on exploiting the correlated log data from the SIEM in order to proficiently identify server threat requests, providing the right support for the SOC investigations to be launched and thus to be efficiently managed. This work also provided a contribution towards the studies of solutions for threat detection systems combining Machine Learning over a Big Data framework, a subject without abundant documentation and with new studies being published every month

to the date of this report. To the extent of the author's knowledge, the combination of these three subjects (IDS, Machine Learning and Big Data) with a fourth, the SIEM, accounts for only one published relevant work. Another contribution of this work was the usage of a real-life dataset from the financial service server requests with all the constraints, challenges, and complexity associated from it, instead of a synthetically generated dataset as most investigation works use, such as the SAIKON 2006 IARPA. The contribution does not come with its shortcomings, as the usage of different datasets from previous studies makes the solution hard to be benchmarked against.

During the development of this work, the main difficulties encountered revolved primarily around the inexperience over cybersecurity, where the SOC team and an extensive investigation played an important role. Another issue that launched a lot of thought about was the methodology used to handle the data imbalance, where the guidance of the dissertation's advisor and investigation work helped to define a fit approach for the use case. Another major difficulty encountered was the shortage of scientific documentation that could fit at the same time intrusion detection, Machine Learning, Big Data and SIEM in the same frame. A third relevant difficulty encountered provided priceless technical growth, the understanding, installation and implementation of whole distributed processing infrastructure services. Another source of issues was the Spark code development using the documentation provided by Spark regarding small details that were important to understand. For the current versions of Spark 2.3 and 2.4, the existence of two different Machine Learning oriented packages, one for RDD (with MLLIB) and another for ML (DataFrames) where the first is being disinvested in favour of the second, currently leaves the whole Machine Learning documentation in a limbo where part of the relevant explanations are in one package and the other in another.

The development of this dissertation work provided the unique opportunity for the author's growth at a theoretical and technical level for both at academic and professional levels.

7. LIMITATIONS AND RECOMMENDATIONS FOR FUTURE WORKS

In this chapter, the limitations and boundaries of this work are described, along with a description of potential future implementations that would add value to this work. The first relevant limitation of this work is related to how the target variable was crafted. As described during the work, the target variable is generated from two sources, manually identified server request threats by the SOC team, and vulnerability assessments from an external and certified company. The second source is a double-edged sword between the benefits of having priceless information of an always updated intensive list of controlled server threat attempts, and a considerable number of non-usable important features and potential methodologies due to it. The vulnerability assessments are executed with a fixed periodicity, in a batch of independent server requests of the service, and from a fixed range of IP addresses. The first consequence is the non-usage of all information regarding IP address source requests, as they all come from the same range, country, city. The second, the non-usage of time-related features, as they are all executed with a fixed periodicity. Lastly, the fixed periodicity and batch mode execution, would irreversible bias the learning algorithms training if aggregation data methodologies were used, capable of representing a small pattern behavior of users over time (for example: aggregated server requests events by minute, by IP address).

The second limitation of the work is related to the elapsed time for loading, preprocessing and training the learning algorithms. The cluster constructed and used for this work is far from presenting the specifications of a cluster in a production environment, where each machine can possess a high number of available cores and a superior available amount of RAM (for example 20 cores and 60GB RAM). Therefore, the usage of a distributed processing through Spark with YARN comes with a price, an overhead in the resources consumed by the YARN itself, the launched AM and all the management processes around it. Therefore, the values achieved are naturally bounded to the experimental setup used.

A third limitation of the work is related to a real production environment constraint, the assumptions that the data was stationary in time. In a production environment, the retraining periodicity of the chosen classifier, the training data retention and time moving window, are aspects that must be attended.

With this said, for future works, one that would add an important contribution would be addition of a system capable answering the previously presented production environment limitation, dealing with the retraining of the chosen classifier over a moving time window period. Another interesting addiction noted for this work would be, for this problem and dataset, the benchmarking of different approaches for handling the imbalanced dataset problem, as well as other learning algorithms. This work would also greatly benefit from a complementary work focused on a more distributed processing-oriented development, exploring different processing times associated with different cluster configurations, either from machines, services, technologies used, among many more.

8. BIBLIOGRAPHY

Akbani, R., Kwek, S., & Japkowicz, N. (2004). Applying Support Vector Machines to Imbalanced Data Sets. In *Lecture Notes in Computer Science* (pp. 39–50). https://doi.org/10.1007/978-3-540-30115-8

Alhawamdeh, M. A. A. (2017). Developing a Conceptual National Information Sharing Security Framework to Combat Cybercrimes in Jordan. In *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing, CSCloud 2017* (pp. 344–350). https://doi.org/10.1109/CSCloud.2017.57

Apache Software Foundation. (2015). HDFS Commands Guide. Retrieved August 19, 2018, from https://hadoop.apache.org/docs/r2.7.1/hadoop-project-dist/hadoop-hdfs/HDFSCommands.html

Apache Software Foundation. (2018a). Apache Hadoop. Retrieved September 23, 2018, from http://hadoop.apache.org/

Apache Software Foundation. (2018b). Apache Hadoop YARN. Retrieved August 14, 2018, from https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

Apache Software Foundation. (2018c). Apache ORC. Retrieved September 9, 2018, from https://orc.apache.org/

Apache Software Foundation. (2018d). Apache Parquet. Retrieved September 9, 2018, from https://parquet.apache.org/

Apache Software Foundation. (2018e). HDFS Architecture. Retrieved August 10, 2018, from http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

Apache Software Foundation. (2018f). MapReduce Tutorial. Retrieved from https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Distributing_Libraries

Apache Software Foundation. (2018g). Spark Python API Docs. Retrieved August 20, 2018, from https://spark.apache.org/docs/2.3.0/api/python/

Apache Spark. (n.d.-a). Linear Methods - RDD-based API. Retrieved August 30, 2018, from https://spark.apache.org/docs/latest/mllib-linear-methods.html#classification

Apache Spark. (n.d.-b). Machine Learning Library (MLlib) Guide. Retrieved November 30, 2018, from https://spark.apache.org/docs/latest/ml-guide.html

Apache Spark. (n.d.-c). Pyspark.ml package. Retrieved May 31, 2018, from https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#module-pyspark.ml.classification

Apache Spark. (n.d.-d). Pyspark.sql module. Retrieved November 21, 2018, from https://spark.apache.org/docs/latest/api/python/pyspark.sql.html

Apache Spark. (n.d.-e). Spark SQL, DataFrames and Datasets Guide. Retrieved from https://spark.apache.org/docs/latest/sql-programming-guide.html

Armbrust, M., Xin, R. S., Lian, C., Huai, Y., Liu, D., Bradley, J. K., … Zaharia, M. (2015). Spark SQL: Relational Data Processing in Spark. In *SIGMOD '15 Proceedings of the 2015 ACM SIGMOD*

International Conference on Management of Data (pp. 1383–1394). https://doi.org/http://dx.doi.org/10.1145/2723372.2742797

Azodi, A., Jaeger, D., Cheng, F., & Meinel, C. (2013). A new approach to building a multi-tier direct access knowledgebase for IDS/SIEM systems. In *2013 IEEE 11th International Conference on Dependable, Autonomic and Secure Computing, DASC 2013* (pp. 118–123). IEEE. https://doi.org/10.1109/DASC.2013.48

Basu, T., & Murthy, C. A. (2012). Effective text classification by a supervised feature selection approach. In *12th IEEE International Conference on Data Mining Workshops (ICDMW 2012)* (pp. 918–925). IEEE. https://doi.org/10.1109/ICDMW.2012.45

Bendovschi, A. (2015). Cyber-Attacks – Trends, Patterns and Security Countermeasures. *Procedia Economics and Finance*, *28*(April), 24–31. https://doi.org/10.1016/S2212-5671(15)01077-1

Bendovschi, A., & Al-Nemrat, A. (2016). Security countermeasures in the cyber-world. In *2016 IEEE International Conference on Cybercrime and Computer Forensic, ICCCF 2016* (pp. 2–8). IEEE. https://doi.org/10.1109/ICCCF.2016.7740440

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *7700 LECTU*, 437–478. https://doi.org/10.1007/978-3-642-35289-8-26

Bernstein, M. N. (2019). *Random Forests*. Retrieved from http://pages.cs.wisc.edu/~matthewb/pages/notes/pdf/ensembles/RandomForests.pdf

Bhowan, U., Johnston, M., & Zhang, M. (2012). Developing New Fitness Functions in Genetic Programming for Classification With Unbalanced Data. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *42*(2), 406–421. https://doi.org/10.1109/TSMCB.2011.2167144

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A Training Algorithm for Optimal Margin Classifiers. In *COLT '92 Proceedings of the fifth annual workshop on Computational learning theory* (pp. 144–152). https://doi.org/10.1145/130385.130401

Bre, F., Gimenez, J. M., & Fachinotti, V. D. (2017). Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks. *Energy and Buildings*, *158*(November), 1–23. https://doi.org/10.1016/j.enbuild.2017.11.045

Breier, J., & Branišová, J. (2017). A Dynamic Rule Creation Based Anomaly Detection Method for Identifying Security Breaches in Log Records. *Wireless Personal Communications*, *94*(3), 497–511. https://doi.org/10.1007/s11277-015-3128-1

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*(2), 123–140. https://doi.org/10.1007/BF00058655

Breiman, L. (2001). Random Forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324

Buczak, A., & Guven, E. (2016). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, *18*(2), 1153–1176. https://doi.org/10.1109/COMST.2015.2494502

Cambridge University Press. (2008a). Extensions to the SVM model. Retrieved September 13, 2018,

from https://nlp.stanford.edu/IR-book/html/htmledition/extensions-to-the-svm-model-1.html

Cambridge University Press. (2008b). Feature selection Chi2 Feature selection. Retrieved May 23, 2018, from https://nlp.stanford.edu/IR-book/html/htmledition/feature-selectionchi2-feature-selection-1.html

Cambridge University Press. (2008c). Soft margin classification. Retrieved September 13, 2018, from https://nlp.stanford.edu/IR-book/html/htmledition/soft-margin-classification-1.html

Center for Strategic and International Studies. (2014). *Net Losses: Estimating the Global Cost of Cybercrime*. *Mcafee*. Retrieved from http://www.mcafee.com/kr/resources/reports/rp-economic-impact-cybercrime2.pdf

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys*, *41*(3), 1–58. https://doi.org/10.1145/1541880.1541882

Chauhan, H., Kumar, V., Pundir, S., & Pilli, E. S. (2013). A Comparative Study of Classification Techniques for Intrusion Detection. In *2013 International Symposium on Computational and Business Intelligence* (pp. 40–43). IEEE. https://doi.org/10.1109/ISCBI.2013.16

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, *16*, 321–357. https://doi.org/10.1613/jair.953

Chen, J., Li, K., Member, S., Tang, Z., & Bilal, K. (2017). A Parallel Random Forest Algorithm for Big Data in a Spark Cloud Computing Environment. *IEEE Transactions on Parallel and Distributed Systems*, *28*(4), 919–933. https://doi.org/10.1109/TPDS.2016.2603511

Cieslak, D. A., Chawla, N. V., & Striegel, A. (2006). Combating imbalance in network intrusion datasets. In *2006 IEEE International Conference on Granular Computing* (pp. 732–737). IEEE. https://doi.org/10.1109/GRC.2006.1635905

Davis, J., & Goadrich, M. (2006). The relationship between Precision-Recall and ROC curves. In *Proceedings of the 23rd international conference on Machine learning - ICML '06* (pp. 233–240). https://doi.org/10.1145/1143844.1143874

Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified Data Processing on Large Clusters. In *OSDI '04: 6th Symposium on Operating Systems Design and Implementation* (pp. 137–149).

Dey, S. (2018). Implementing a Soft-Margin Kernelized Support Vector Machine Binary Classifier with Quadratic Programming in R and Python. Retrieved November 23, 2018, from https://www.datasciencecentral.com/profiles/blogs/implementing-a-soft-margin-kernelized-support-vector-machine

Dolev, S., Elovici, Y., Kesselman, A., & Zilberman, P. (2009). Trawling traffic under attack: Overcoming DDoS attacks by target-controlled traffic filtering. In *2009 International Conference on Parallel and Distributed Computing, Applications and Technologies, PDCAT Proceedings* (pp. 336–341). IEEE. https://doi.org/10.1109/PDCAT.2009.40

Epishkina, A., & Zapechnikov, S. (2016). A Syllabus on Data Mining and Machine Learning with Applications to Cybersecurity. In *2016 Third International Conference on Digital Information Processing, Data Mining, and Wireless Communications (DIPDMWC)* (pp. 194–199). IEEE. https://doi.org/10.1109/DIPDMWC.2016.7529388

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). From data mining to knowledge discovery in

databases. *AI Magazine*, *17*(3), 37–54. https://doi.org/10.1609/aimag.v17i3.1230

Fitriani, S., Mandala, S., & Murti, M. A. (2016). Review of semi-supervised method for Intrusion Detection System. In *2016 Asia Pacific Conference on Multimedia and Broadcasting (APMediaCast)* (pp. 36–41). https://doi.org/10.1109/APMediaCast.2016.7878168

Forman, G. (2003). An Extensive Empirical Study of Feature Selection Metrics for Text Classification. *Journal of Machine Learning Research*, *3*, 1289–1305.

Fu, J., Sun, J., & Wang, K. (2016). SPARK – A Big Data Processing Platform for Machine Learning. In *2016 International Conference on Industrial Informatics - Computing Technology, Intelligent Technology, Industrial Information Integration* (pp. 48–51). IEEE. https://doi.org/10.1109/ICIICII.2016.27

Galar, M., Fernández, A., Barrenechea, E., Bustince, H., & Herrera, F. (2012). A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *42*(4), 463–484. https://doi.org/10.1109/TSMCC.2011.2161285

Ghemawat, S., Gobioff, H., & Leung, S. (2003). The Google File System. In *SOSP '03 Proceedings of the nineteenth ACM symposium on Operating systems principles* (pp. 29–43). ACM.

Grover, M., & Malaska, T. (2016). *Top 5 Mistakes When Writing Spark Applications*. Spark Summit 2016. Retrieved from https://databricks.com/session/top-5-mistakes-when-writing-spark-applications

Gupta, G. P., & Kulariya, M. (2016). A Framework for Fast and Efficient Cyber Security Network Intrusion Detection Using Apache Spark. *Procedia Computer Science*, *93*(September), 824–831. https://doi.org/10.1016/j.procs.2016.07.238

Hakim, L., Sartono, B., & Saefuddin, A. (2017). Bagging Based Ensemble Classification Method on Imbalance Datasets. *IJCSN -International Journal of Computer Science and Network*, *6*(6), 670–676. Retrieved from http://ijcsn.org/IJCSN-2017/6-6/Bagging-Based-Ensemble-Classification-Method-on-Imbalance-Datasets.pdf

He, S., Zhu, J., He, P., & Lyu, M. R. (2016). Experience Report: System Log Analysis for Anomaly Detection. In *Proceedings International Symposium on Software Reliability Engineering, ISSRE* (pp. 207–218). IEEE. https://doi.org/10.1109/ISSRE.2016.21

Hoque, N., Bhuyan, M. H., Baishya, R. C., Bhattacharyya, D. K., & Kalita, J. K. (2014). Network attacks: Taxonomy, tools and systems. *Journal of Network and Computer Applications*, *40*, 307–324. Retrieved from https://doi.org/10.1016/j.jnca.2013.08.001

Hui, K.-L., Kim, S. H., & Wang, Q.-H. (2017). Cybercrime deterrence and international legislation: Evidence from distributed denial of service attacks. *MIS Quarterly*, *41*(2), 497–523.

Hunt, T. (2019). Have i been pwned? Retrieved February 18, 2019, from https://haveibeenpwned.com/

IBM. (2011). *IBM SPSS Modeler CRISP-DM Guide*. IBM Corporation.

Internet Crime Complaint Center. (2016). *2016 Internet Crime Report*.

Jamali, I., Bazmara, M., & Jafari, S. (2012). Feature Selection in Imbalance data sets. *International Journal of Computer Science Issues (IJCSI)*, *9*(3), 42–45.

Jenab, K., & Moslehpour, S. (2016). Cyber Security Management: A Review. *Business Management Dynamics*, *5*(11), 16–39.

Joglekar, P., & Pise, N. (2016). Solving Cyber Security Challenges using Big Data. *International Journal of Computer Applications*, *154*(4), 9–12. Retrieved from https://pdfs.semanticscholar.org/b9aa/3fe200c8e6087e13181969b03c4a6d7ae570.pdf

Joseph, A. D., Laskov, P., Roli, F., Tygar, J. D., & Nelson, B. (2012). *Machine Learning Methods for Computer Security*. *Dagstuhl Reports* (Vol. 2). https://doi.org/10.4230/DagRep.2.9.109

K, A. A., Aljahdali, S., & Hussain, S. N. (2013). Comparative Prediction Performance with Support Vector Machine and Random Forest Classification Techniques. *International Journal of Computer Applications*, *69*(11), 12–16.

Kawa, A. (2014). Introduction to YARN. Retrieved August 20, 2018, from https://developer.ibm.com/tutorials/bd-yarn-intro/

Kawakubo, H., & Yoshida, H. (2012). Rapid Feature Selection Based on Random Forests for High-Dimensional Data. *Information Processing Society of Japan*, *2012-NaN-8*(3), 1–7.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *14th international joint conference on Artificial intelligence* (Vol. 2, pp. 1137–1143).

Kulariya, M., Saraf, P., Ranjan, R., & Gupta, G. P. (2016). Performance Analysis of Network Intrusion Detection Schemes using Apache Spark. In *2016 International Conference on Communication and Signal Processing (ICCSP)* (pp. 1973–1977). IEEE. https://doi.org/10.1109/ICCSP.2016.7754517

Kumar, S. R., Yadav, S. A., Sharma, S., & Singh, A. (2016). Recommendations for effective cyber security execution. In *2016 1st International Conference on Innovation and Challenges in Cyber Security, ICICCS 2016* (pp. 342–346). IEEE. https://doi.org/10.1109/ICICCS.2016.7542327

Lardeux, C., Frison, P., Tison, C., Souyris, J., Stoll, B., Fruneau, B., & Rudant, J.-P. (2009). Support Vector Machine for Multifrequency SAR Polarimetric Data Classification. In *IEEE Transactions on Geoscience and Remote Sensing* (pp. 4143–4152). IEEE. https://doi.org/10.1109/IGARSS.2006.131

Lee, J., Kim, Y. S., Kim, J. H., & Kim, I. K. (2017). Toward the SIEM Architecture for Cloud-based Security Services. In *2017 IEEE Conference on Communications and Network Security (CNS)* (pp. 398–399). IEEE. https://doi.org/10.1109/CNS.2017.8228696

Li, C. (2007). Classifying imbalanced data using a bagging ensemble variation (BEV). In *Proceedings of the 45th Annual Southeast Regional Conference* (pp. 203–208). https://doi.org/10.1145/1233341.1233378

López, V., Fernández, A., García, S., Palade, V., & Herrera, F. (2013). An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Information Sciences*, *250*, 113–141. https://doi.org/10.1016/j.ins.2013.07.007

Louppe, G., Wehenkel, L., Sutera, A., & Geurts, P. (2013). Understanding variable importances in Forests of randomized trees. In *Advances in neural information processing systems* (pp. 1–9).

Luettmann, B. M., & Bender, A. C. (2007). Man-in-the-middle attacks on auto-updating software. *Bell Labs Technical Journal*, *12*(3), 131–138. https://doi.org/10.1002/bltj

Mahmood, T., & Afzal, U. (2014). Security Analytics: Big Data Analytics for Cybersecurity. In *2013 2nd National Conference on Information Assurance (NCIA)* (pp. 129–134). https://doi.org/10.1109/NCIA.2013.6725337

Martorella, C. (n.d.). A fresh new look into information gathering. https://doi.org/10.1017/CBO9781107415324.004

Masters, D., & Luschi, C. (2018). Revisiting Small Batch Training for Deep Neural Networks, 1–18. Retrieved from http://arxiv.org/abs/1804.07612

Mathews;, B., & Aasim, O. (2018). Hadoop MapReduce Tutorial. Retrieved October 12, 2018, from https://www.dezyre.com/hadoop-tutorial/hadoop-mapreduce-tutorial-

Mitchell, T. M. (1997). *Machine learning*. Boston, Burr Ridge, Dubuque, Madison, New York, San Francisco, St. Louis: WCB/Mcgraw-Hill.

Murphy, K. P. (2012). *Machine learning: A probabilistic perspective*. Massachusetts: The MIT Press.

Nadiammai, G. V., & Hemalatha, M. (2012). Perspective analysis of machine learning algorithms for detecting network intrusions. In *2012 Third International Conference on Computing, Communication and Networking Technologies (ICCCNT'12)* (pp. 1–7). IEEE. https://doi.org/10.1109/ICCCNT.2012.6395949

Nagle, M. K., & Chaturvedi, S. K. (2013). Feature Extraction Based Classification Technique for Intrusion Detection System. *International Journal of Engineering Research and Development*, *8*(2), 23–38.

Nair, L. R., Shetty, S. D., & Shetty, S. D. (2017). Applying spark based machine learning model on streaming big data for health status prediction. *Computers and Electrical Engineering*, *0*, 1–7. https://doi.org/10.1016/j.compeleceng.2017.03.009

Neustar®. (2012). *DDoS Survey: Q1 2012 When Businesses Go Dark*. Retrieved from http://hello.neustar.biz/rs/neustarinc/images/neustar-insights-ddos-attack-survey-q1-2012.pdf

Ng, A. (2018). *Supervised learning* (No. CS229 Lecture notes).

Ng, R. (2018). Logistic Regression. Retrieved August 27, 2018, from https://www.ritchieng.com/logistic-regression

Nguyen, C., Wang, Y., & Nguyen, H. N. (2013). Random forest classifier combined with feature selection for breast cancer diagnosis and prognostic. *Journal of Biomedical Science and Engineering*, *6*(5), 551–560. https://doi.org/http://dx.doi.org/10.4236/jbise.2013.65070

Nikolskaya, K. Y., Ivanov, S. A., Golodov, V. A., Minbaleev, A. V., & Asyaev, G. D. (2017). Review of modern DDoS-attacks, methods and means of counteraction. In *Proceedings of the 2017 International Conference "Quality Management, Transport and Information Security, Information Technologies", IT and QM and IS 2017* (pp. 87–89). https://doi.org/10.1109/ITMQIS.2017.8085769

Nykodym, T., Kraljevic, T., Wang, A., & Wong, W. (2019). *Generalized Linear Modeling with H2O*. (A. Bartz, Ed.) (7th ed.). Mountain View, CA: H2O.ai, Inc.

Peña, I. A. de la. (2017). *Fraud detection in online payments using Spark ML*. KTH Royal Institute of Technology.

Platt, J. C. (2000). Probabilities for SV Machines. In A. J. Smola, P. L. Bartlett, B. Scholkopf, & D.

Schuurmans (Eds.), *Advances in Large Margin Classifiers* (pp. 61–74). MIT Press.

Prachi. (2016). Usage of Machine Learning for Intrusion Detection in a Network. *International Journal of Computer Networks And Applications*, *3*(6), 139–147. https://doi.org/10.22247/ijcna/2016/41278

Quick, M., Hollowood, E., Miles, C., & Hampson, D. (2017). World's Biggest Data Breaches. Retrieved January 14, 2018, from http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/

Rajan, A. V., Ravikumar, R., & Shaer, M. Al. (2017). UAE cybercrime law and cybercrimes - An analysis. In *2017 International Conference on Cyber Security And Protection Of Digital Services, Cyber Security 2017* (pp. 1–6). IEEE. https://doi.org/10.1109/CyberSecPODS.2017.8074858

Saad, A., Amran, A. R., Afif, I. I., Zolkeple, A. H., Said, A. I. A., Hamzah, M. F., & Salim, W. N. S. W. (2016). Privacy and security gaps in mitigating Cyber crime: The review. In *2nd International Symposium on Agent, Multi-Agent Systems and Robotics, ISAMSR 2016* (pp. 92–99). IEEE. https://doi.org/10.1109/ISAMSR.2016.7810009

Singh, A., & Purohit, A. (2015). A survey on methods for solving data imbalance problem for classification. *International Journal of Computer Applications*, *127*(15), 37–41.

Sisiaridis, D., & Markowitch, O. (2017). Feature Extraction and Feature Selection: Reducing Data Complexity With Apache Spark. *International Journal of Network Security & Its Applications (IJNSA)*, *9*(6), 39–51. https://doi.org/10.5121/ijnsa.2017.9604

Suen, J. (2017). Spark-stratifier. Retrieved August 17, 2018, from https://github.com/interviewstreet/spark-stratifier

Suh-Lee, C., Jo, J.-Y., & Kim, Y. (2016). Text mining for security threat detection: Discovering hidden information in unstructured log messages. In *2016 IEEE Conference on Communications and Network Security (CNS 2016)* (pp. 252–260). IEEE. https://doi.org/10.1109/CNS.2016.7860492

The Department of Commerce Internet Policy Task Force. (2011). *Cybersecurity, Innovation and the Internet Economy*. U.S. Department of Commerce. Retrieved from papers3://publication/uuid/2DE4A620-537A-41D3-8CA8-DCA889CADE56

The OSI Model - Features, Principles and Layers. (2018). Retrieved from https://www.studytonight.com/computer-networks/complete-osi-model

Timčenko, V., & Gajin, S. (2017). Ensemble classifiers for supervised anomaly based network intrusion detection. In *Proceedings 2017 IEEE 13th International Conference on Intelligent Computer Communication and Processing, ICCP 2017* (pp. 13–19). IEEE. https://doi.org/10.1109/ICCP.2017.8116977

Ussath, M., Jaeger, D., Cheng, F., & Meinel, C. (2017). Identifying Suspicious User Behavior with Neural Networks. In *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing* (pp. 255–263). IEEE. https://doi.org/10.1109/CSCloud.2017.10

Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., … Baldeschwieler, E. (2013). Apache Hadoop YARN : Yet Another Resource Negotiator. In *SOCC '13 Proceedings of the 4th annual Symposium on Cloud Computing* (pp. 1–16). ACM. https://doi.org/10.1145/2523616.2523633

Verizon. (2017). *2017 Data Breach Investigations Report*. *Verizon Business Journal*.

https://doi.org/10.1017/CBO9781107415324.004

Viegas, E. K., Santin, A. O., & Oliveira, L. S. (2017). Toward a reliable anomaly-based intrusion detection in real-world environments. *Computer Networks*, *127*, 200–216. https://doi.org/10.1016/j.comnet.2017.08.013

Wang, L., & Jones, R. (2017). Big Data Analytics for Network Intrusion Detection: A Survey. *International Journal of Networks and Communications*, *7*(1), 24–31. https://doi.org/10.5923/j.ijnc.20170701.03

Wisesa, H. A., Ma'sum, M. A., Mursanto, P., & Febrian, A. (2016). Processing Big Data with Decision Trees: A Case Study in Large Traffic Data. In *2016 International Workshop on Big Data and Information Security (IWBIS)* (pp. 115–120). IEEE. https://doi.org/10.1109/IWBIS.2016.7872899

World Economic Forum. (2017). *The Global Risks Report 2017*.

Wu, J. S., Lee, Y. J., Wei, T. E., Hsieh, C. H., & Lai, C. M. (2017). ChainSpot: Mining service logs for cyber security threat detection. In *2016 IEEE Trustcom/BigDataSE/ISPA* (pp. 1867–1874). IEEE. https://doi.org/10.1109/TrustCom.2016.0286

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., Mccauley, M., … Stoica, I. (2012). Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *NSDI'12 Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation* (pp. 1–14). USENIX Association Berkeley, CA, USA.

9. APPENDIX

9.1. DATA MINING TASKS

Fayyad et al. (1996) define six of the most common Data Mining tasks: i) Association rule learning; ii) Clustering; iii) Classification and Regression; iv) Anomaly Detection; and v) Summarization.

i. **Association rule learning** – Also referenced in literature as dependency modelling, is a Data Mining task focused on the identification of event relationships or patterns that describe subsets of data. Often used for commercial and marketing purposes through the study of customer purchase habits and their product acquisition relationships (Fayyad et al., 1996).
ii. **Clustering** - Is a task focused on grouping together observations (Fayyad et al., 1996). Each observation in a group (cluster) should be similar to each other, based on some of the observation's attributes, and dissimilar to the other observations in the other clusters. Each cluster should be homogeneous or compact and every observation in them should have similar behaviour (Epishkina & Zapechnikov, 2016).
iii. **Classification and Regression** - Are tasks focused on the generalization of a known structure of data to new data (Fayyad et al., 1996). In other words, given a set of training examples, new observations can be mapped to class or label variable (classification) or real-valued prediction variable (regression) (Fayyad et al., 1996).
iv. **Anomaly Detection** – Commonly referred as an outlier, change or deviation detection, is a set of Data Mining techniques focused on the detection and identification of unusual data records that deviate or are considered unfit of what was previously validated as a normal pattern value (Fayyad et al., 1996). The set of techniques can either be used as supervised, unsupervised or even semi-supervised learning algorithms (Chandola, Banerjee, & Kumar, 2009). The inherently unbalanced datasets used for these tasks, where the ratio between normal events and abnormal ones is disproportionate, is what differentiates the methods from the previous ones presented such as clustering or classification (Epishkina & Zapechnikov, 2016).
v. **Summarization** – Is a set of varied methods focused on the compact description and representation of data, often applied in the process of interactive exploratory analysis, result visualization or even report generation. The statistical measures of average and standard deviation are commonly used for the description of numerical attributes, while other methods focus on the discovery relationships between variables using multivariate visualization techniques (Fayyad et al., 1996).

9.2. HDFS TECHNOLOGY AND ARCHITECTURE OVERVIEW

The HDFS supports large files using primarily in batch processing under the premises that a "computation requested by an application is much more efficient if it is executed near the data it operates on" (Apache Software Foundation, 2018e). Therefore, the HDFS "minimizes network congestion and increases the overall throughput of the system" (Apache Software Foundation, 2018e) by providing "interfaces for applications to move themselves closer to where the data is located" rather than making the data reach the processing machine (Apache Software Foundation, 2018e). The HDFS also provides fault tolerance to the distributed data storage through redundancy, fault detection and recovery (Apache Software Foundation, 2018e).

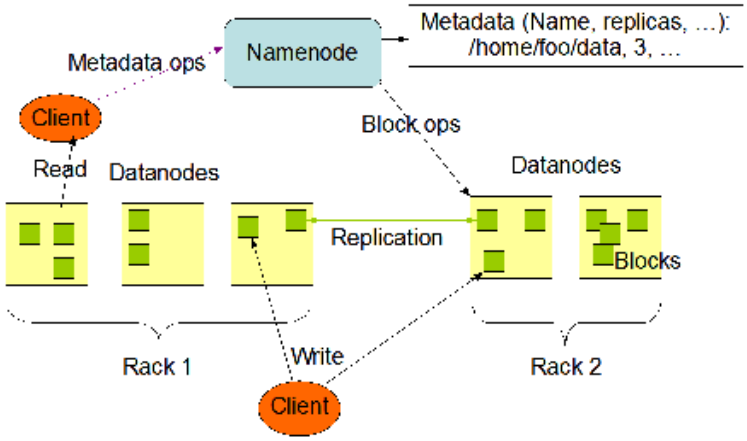


Figure 9.1 - HDFS Architecture (Apache Software Foundation, 2018e)

The HDFS architecture is comprised of a NameNode (master node) and multiple DataNodes (slave nodes), as represented on Figure 9.1 (Apache Software Foundation, 2018e). The NameNode is responsible for mapping the blocks across all DataNodes and performs storage and management of the access of HDFS metadata across all the machines (Apache Software Foundation, 2018e). The DataNodes are the machines where the HDFS data is stored and are responsible for executing read and write requests from the HDFS clients as well as "block creation, deletion, and replication upon instruction from the NameNode" (Apache Software Foundation, 2018e). Typically, a file is stored in a sequence of mapped blocks across the machines of the cluster and is replicated for fault tolerance across different DataNodes. Every stored block is mapped through the metadata stored in the NameNode. Periodically the Namenode receives a status report, through Heartbeat and a Blockreport, regarding the availability of the blocks in the machines and proceeds to execute replication tasks of the unavailable blocks according to an indicated replication factor (Apache Software Foundation, 2018e).

9.3. YARN TECHNOLOGY AND ARCHITECTURE OVERVIEW

The YARN is designed for resource management and job scheduling of the cluster. As referenced previously, the architecture is comprised by two entities in what is referred to as the "data-computation framework" (Figure 9.2) (Apache Software Foundation, 2018b): i) the RM for global management of the cluster resources among all applications submitted to the system; and ii) NodeManager, an agent present in each DataNode, responsible for controlling the resource containers, their resource consumption monitoring (CPU, memory, disk, network, etc) and respective reporting to the RM/Scheduler (Apache Software Foundation, 2018b; Vavilapalli et al., 2013).

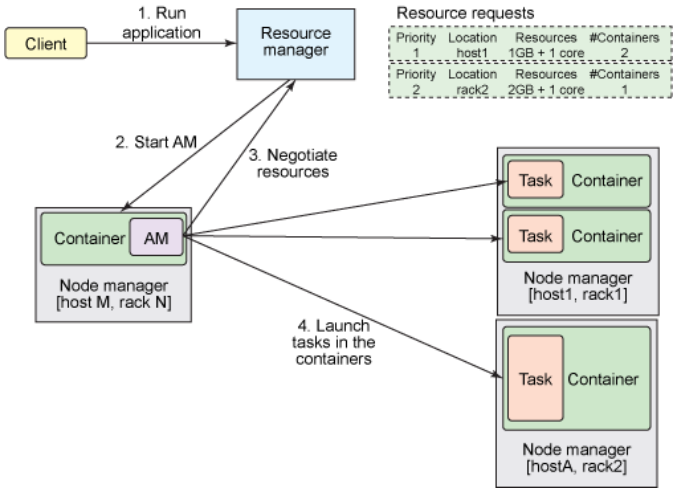


Figure 9.2 - YARN Architecture (Apache Software Foundation, 2018f)

The **RM** is comprised of the Scheduler and the ApplicationManager. The Scheduler is responsible job scheduling activities based on the resource requirements of the applications, as well as the resource allocation itself to the various running applications using the abstract notion of resource container. While the ApplicationsManager is tasked to accept application submissions, perform the negotiation procedures for launching the AM on the first container.

Upon an application submission to the RM, the ApplicationManager validates, accepts and forwards the admitted application to the scheduler. The scheduler will pick it from a queue and contacts a **NodeManager** to start a new container and launch a new AM for the application submitted. The per-application AM will firstly send resource requests to the RM's Scheduler asking for the number of containers needed to run an application's tasks requirements. After the containers are granted, the AM will contact the NodeManagers to use the resources to execute application tasks. The AM will be negotiating containers for all the tasks required until the application execution is completed, monitoring their progress, restarting failed tasks using new containers, and reporting the progress to the client that originally submitted the application. Through the process, the ApplicationManager will monitor the AM health status and upon failure, it will restart the AM in a new container (Apache Software Foundation, 2018b; Kawa, 2014; Vavilapalli et al., 2013).

9.4. MAPREDUCE PROCESS OVERVIEW

The workflow of a MapReduce job starts by **splitting** the input data into fixed-sized independent chunks. In parallel node processes, each chunk is parsed to key/value pairs and passed each pair to the user-defined **map** function to produce a key-value pair output. The outputs of the map tasks are written on the local disk of the respective node in what is considered the intermediate outputs of the whole workflow.

The Reducer phase starts with the reducer nodes reading the previous step's outputs and performing **shuffling** and **sorting** operations in order to group together the same intermediate keys for the same reduce tasks. The sorted results are then aggregated together by unique intermediate key and passed to user-defined reduce functions, in what is called the **reduce** phase, to produce a single output for the job in a key-value format (Apache Software Foundation, 2018f; Dean & Ghemawat, 2008).

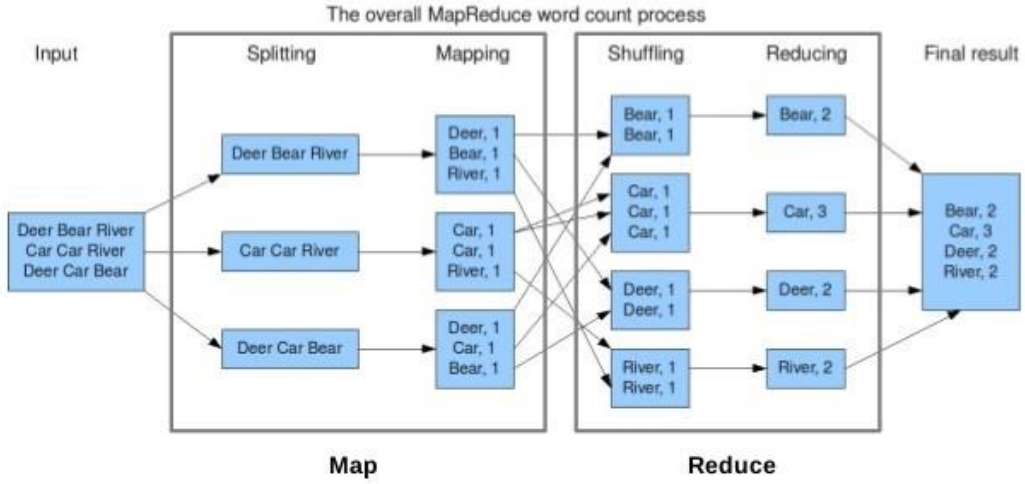


Figure 9.3 - MapReduce Applied to the Word Count Example (Mathews; & Aasim, 2018)

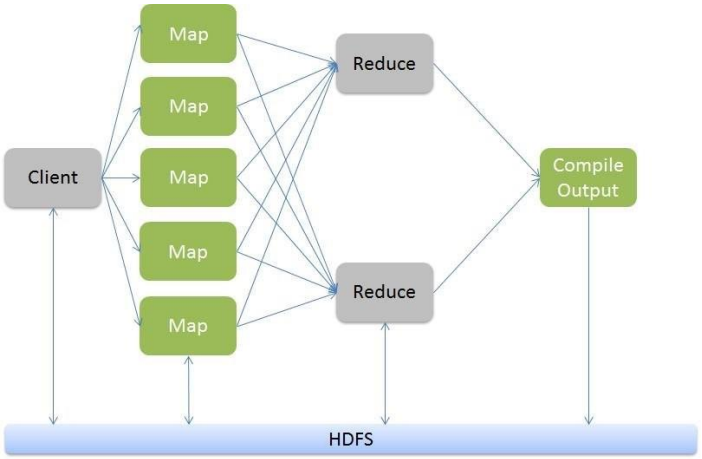


Figure 9.4 - MapReduce Interaction With HDFS (Mathews; & Aasim, 2018)

9.5. EXPERIMENTAL SETUP

An overview of the cluster general architecture regarding the machines, hardware and software used for all the experiments, tests, developments and models of this work is represented on Figure 9.5.

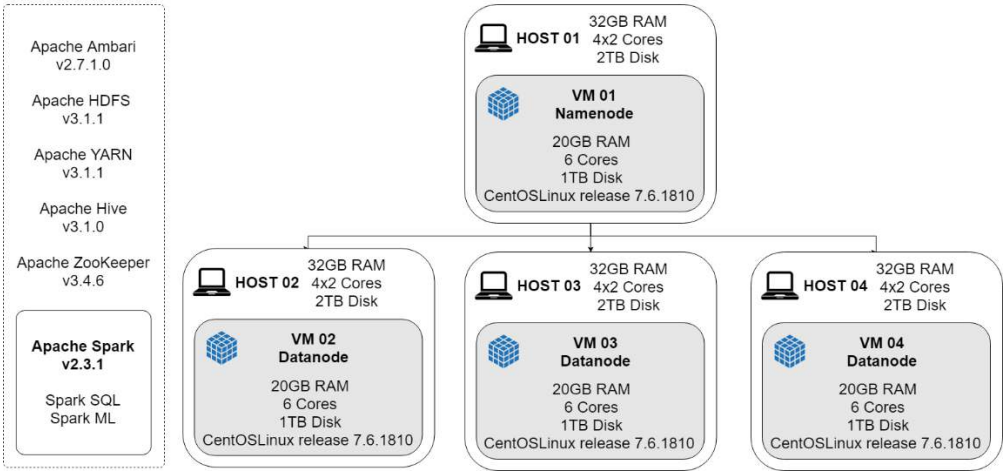


Figure 9.5 - Experimental Cluster Setup Architecture

The full-service stack installed versions for the cluster are indicated and described on Table 9.1.

Table 9.1 - Experimental Cluster Setup Services

Service	Version	Description
Apache Ambari	2.7.1.0	Cluster services management and monitoring
Apache HDFS	3.1.1	Apache Hadoop Distributed File System responsible for the cluster storage
Apache YARN	3.1.1	Cluster resource Manager
Apache Hive	3.1.0	Data warehouse system for ad-hoc queries and analysis of large datasets and table and storage management service
Apache ZooKeeper	3.4.6	Centralized service which provides highly reliable distributed coordination
Apache Spark	2.3.1	Apache Spark 2.3 is a fast and general engine for large-scale data processing

9.6. PLOT REPRESENTATION OF THE INTERVAL FEATURE "BYTES" – ORIGINAL DATASET

For the input interval feature "bytes", a plot representation (histogram) is presented in order to support the summary statistics described during the work on Figure 9.6:

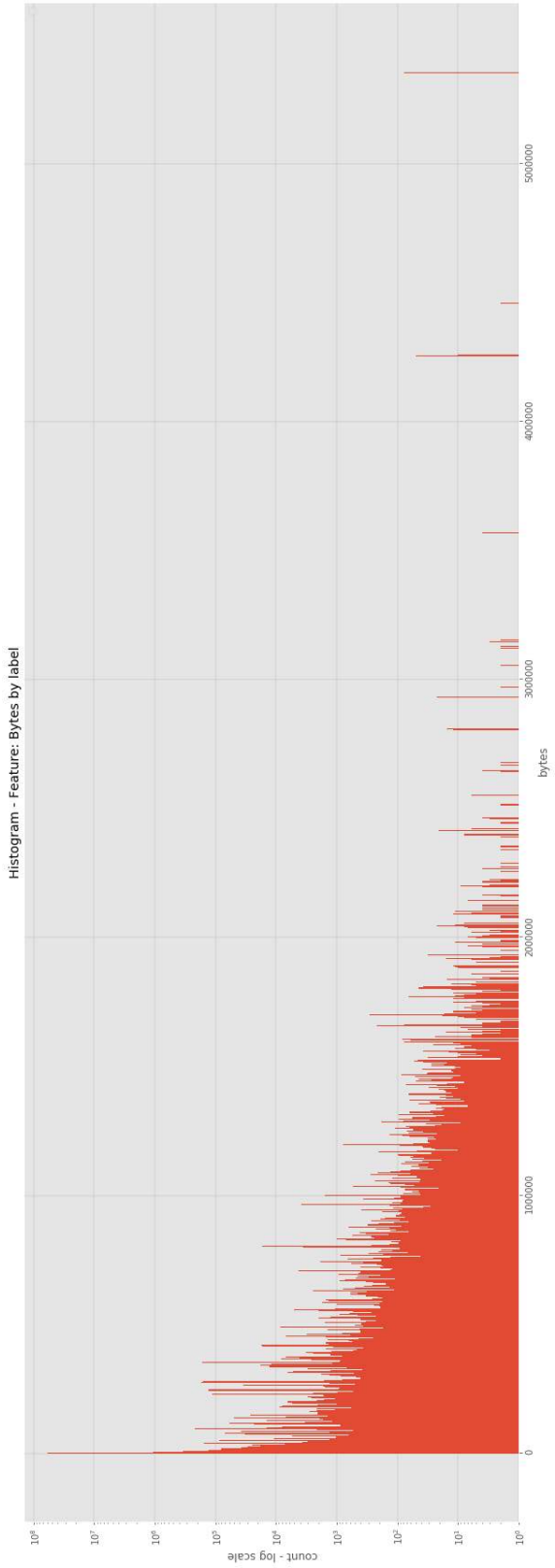


Figure 9.6 - Plot Representation of the Input Interval Feature "bytes"

9.7. SUMMARY STATISTICS NOMINAL FEATURES – TRANSFORMED DATASET

A more detailed summary statistics for the transformed features is presented:

Table 9.2 - Summary Statistics for the Transformed ReferrerContentGrouped

referrerContentGrouped	depvar	Count	Contribution by depvar	Contribution Total
no_msg	0	62235537	99.9995%	99.0624%
with_msg	0	341	0.0005%	0.0005%
no_msg	1	574022	97.5041%	0.9137%
with_msg	1	14694	2.4959%	0.0234%
Total	**-**	**62824594**	**-**	**100.0000%**

Table 9.3 - Summary Statistics for the Transformed RequestClientDeviceGrouped

requestClientDeviceGrouped	depvar	Count	Contribution by depvar	Contribution Total
android	0	36770204	59.0820%	58.5284%
ios	0	22266963	35.7783%	35.4431%
unknown_requests	0	2090432	3.3589%	3.3274%
merchant_other_requests	0	520354	0.8361%	0.8283%
others	0	296734	0.4768%	0.4723%
merchant_programatic_requests	0	286002	0.4595%	0.4552%
desktop	0	4056	0.0065%	0.0065%
not_merchant_bot_requests	0	452	0.0007%	0.0007%
not_merchant_programatic_requests	0	431	0.0007%	0.0007%
tool_requests	0	217	0.0003%	0.0003%
merchant_bot_requests	0	33	0.0001%	0.0001%
unknown_requests	1	541078	91.9082%	0.8613%
desktop	1	29701	5.0450%	0.0473%
ios	1	13425	2.2804%	0.0214%
others	1	2612	0.4437%	0.0042%
tool_requests	1	1900	0.3227%	0.0030%
Total	**-**	**62824594**	**-**	**100.0000%**

Table 9.4 - Summary Statistics for the Transformed RequestMethodGrouped

requestMethodGrouped	depvar	Count	Contribution by depvar	Contribution Total
common_post	0	62184510	99.9175%	98.9812%
uncommon	0	38212	0.0614%	0.0608%
common_get	0	13156	0.0211%	0.0209%
common_get	1	570350	96.8803%	0.9078%
common_post	1	12601	2.1404%	0.0201%
uncommon	1	5765	0.9792%	0.0092%
Total	**-**	**62824594**	**-**	**100.0000%**

Table 9.5 - Summary Statistics for the Transformed httpCodeGrouped

httpCodeGrouped	depvar	Count	Contribution by depvar	Contribution Total
2xx	0	62023227	99.6583%	98.7244%
4xx	0	188209	0.3024%	0.2996%
5xx	0	20384	0.0328%	0.0324%
3xx	0	4058	0.0065%	0.0065%
4xx	1	572710	97.2812%	0.9116%
3xx	1	13842	2.3512%	0.0220%
2xx	1	2163	0.3674%	0.0034%
5xx	1	1	0.0002%	0.0000%
Total	**-**	**62824594**	**-**	**100.0000%**

Table 9.6 - Summary Statistics for the Transformed RequestUrlFileNameGrouped

requestUrlFileNameGrouped	depvar	Count	Contribution by depvar	Contribution Total
urlFile_noExt	0	62225977	99.98409%	99.04716%
urlFolder	0	4962	0.00797%	0.00790%
urlFile_wExt_notImage	0	4909	0.00789%	0.00781%
urlFile_wExt_image	0	30	0.00005%	0.00005%
urlFolder	1	285303	48.46191%	0.45413%
urlFile_wExt_notImage	1	188199	31.96771%	0.29956%
urlFile_noExt	1	106521	18.09378%	0.16955%
urlFile_wExt_image	1	8693	1.47660%	0.01384%
Total	**-**	**62824594**	**-**	**100.0000%**

9.8. IMBALANCED DATASET HANDLING – EXTERNAL OR DATA LEVEL APPROACHES

Three approach groups of methods are documented for the external approaches for handling imbalanced datasets (Li, 2007): i) Sampling methods; ii) Bagging-based methods; and iii) Boosting-based methods

The **sampling methods** are techniques that seek to change the training set in order to balance it for the learning algorithm. The undersampling and oversampling are sets of methods focused on reducing the majority class or increase minority class, respectively (Singh & Purohit, 2015). Both techniques have shown improved performances over imbalanced dataset (Singh & Purohit, 2015). While the undersampling reduces the training time of the learning algorithm, it takes the risk of discarding potential useful information from the majority class instances. On the other hand, the oversampling techniques increase the training set in order to bring balance to the class representation, however it contributes to penalizing the training time of the learning algorithm, which can incur in aggravated processing times when facing data volumes such as the ones present in Big Data (Singh & Purohit, 2015). Another problem of the oversampling techniques is related to the method chosen to generate the synthetic data. If for one side,

repeatedly increasing the minority class representation using copies of the already existing instances can lead to overfitting of the class, the generation of synthetic new instances of the minority class can lead to the generation of unnecessary and unrealistic samples that can lead to the distortion of the business problem to be solved (Singh & Purohit, 2015). A series of different approaches have been used in several types of research, such as Under Sampling, Over Sampling, Synthetic Minority Over-sampling Technique (SMOTE), among many others (Singh & Purohit, 2015).

The **bagging-based methods**, belong to the group of ensemble methods where sampling techniques are repeatedly applied with replacement on the original imbalanced dataset to produce multiple balanced datasets (Galar et al., 2012; Hakim et al., 2017). For each dataset generated a learning algorithm is trained for classification or regression, depending on the case (Hakim et al., 2017). The final output is the aggregated combination of each trained balanced dataset, either through majority voting or by averaging the results, respectively for classification or regression (Hakim et al., 2017). Bagging methods have been reported to produce increased performances over different imbalanced datasets (Hakim et al., 2017). A series of different approaches have been used in several types of research, such as UnderBagging, OverBagging, among many others where it is included the method used for this work, the BEV (Hakim et al., 2017).

The **boosting-based methods** that also belong to the group of ensemble methods. The AdaBoost algorithm is the most referenced boosting algorithm and works by training multiple classifiers serially, assigning on each iteration increasing weights on misclassified instances and decreasing weights on the correctly classified instances in order to produce a set of focused and diverse classifiers (Galar et al., 2012). Furthermore, for each classifier produced, a new weight is assigned according to the respective learner performance, each partially contributing for the final output of the algorithm, through a weighted majority voting (Galar et al., 2012). Some of the most referenced methods for handling imbalanced data are based on different variations of the AdaBoost algorithm and work by manipulating on every iteration the "weight distribution used to train the next classifier toward the minority class" (Galar et al., 2012). The manipulation of the training set for the next classifier on each iteration is processed through several variations of sampling techniques, such as oversampling through generation of synthetic data, undersampling the majority class, among many others, in order to train classifiers with more balanced representations of the target classes (Galar et al., 2012). According to the work presented by Galar et al. (2012), a series of different approaches have been used in several types of research, such as the SMOTEBoost, MSMOTEBoost, RUSBoost, and DataBoost-IM algorithms.

9.9. ARTIFICIAL NEURAL NETWORKS

Inspired by the biological learning systems used by the human brain, where a complex interconnected network of neurons is used for information-processing activities, the ANN were created on the idea of reproducing the highly parallel computation process that neurons underwent (Buczak & Guven, 2016; Mitchell, 1997). Starting with the concept of a system based on a single unit called perceptron capable of producing a hyperplane decision surface, several developments over the years have evolved the ANN systems into an algorithm robust to noise in the training data and successfully applied in a wide variety of fields (Mitchell, 1997).

The authors Ussath, Jaeger, Cheng, and Meinel (2017) provide an intuitive perceptive on how general ANN process the information to make predictions: In a network, for each connected pair of neurons, a weight is associated. The connected neurons are able to weight the contribution of the features fed to the network so that results can be correctly predicted.

According to Mitchell (1997), the most common learning model used to train multilayer feedforward networks are based on the backpropagation algorithm. This training algorithm learns the weights and adjusts them using, among others, ruled gradient descent-based approaches to minimize the error between the produced output and the target output values (Mitchell, 1997). Considering an error surface associated to the hypothesis space of all weight vectors, the algorithm iteratively adjusts the weights for each training instance, by searching for the weight vector that produces the steepest descent along the error surface in an attempt to converge for the global minimum error (Mitchell, 1997). Subsequently, the trained model will be able to predict results over new and unseen instances (Ussath et al., 2017).

Among the range of architectures of neural networks, the feedforward neural networks architecture is considered to be the least complex (Ussath et al., 2017). The network is comprised of multiple layers of neurons, where all the neurons of each layer are connected to all the neurons of the following layer and no connections are established between neurons of the same layers. Three types of layers are used: i) input layer; ii) output layer; and iii) hidden layers.

The first one, the **input layer**, is composed of the neurons that accept input values, in other words, the inputs from the features of each instance. The second one, the **output layer** is the final layer of the network and returns the result of training or predicting an instance. The third layer, the **hidden layers,** are optional and located between the input and output layers and allow the model to solve non-linear problems (Buczak & Guven, 2016; Mitchell, 1997; Ussath et al., 2017).

A feed-forward network with a fixed number of input units, hidden units and output units, is initialized with the connections between neurons weighted randomly. For each training example, the inputs units are

propagated forward through the network until it computes the output results of every unit in the output layer. Each unit takes a vector of input values from the previous layer, the weights of the connections, and a bias term to create a linear combination of the inputs and applies an activation function to produce an output. The output results of the units in the output layer are compared with the target values of the training example, and from the comparison, an error measure is generated. The errors metric from the output units are then propagated and calculated backwards through the network updating each of the network weights (Mitchell, 1997; Ussath et al., 2017).

Mitchel (1997) describes ANN learning methods as robust to noise in the training data to some extent. One interesting property of the hidden unit layers is the potential definition of new hidden layer features relevant for learning the target function generated from properties of the training examples (Mitchell, 1997).

According to Mitchel (1997), two of the downsides of this algorithms are the long training times and the human interpretation of the weights learnt by the algorithm when compared with other algorithms such as decision trees. Another aspect of relevance is the number of iterations used during the execution. As the complexity of the learned decision surface increases with the number of iterations so does the accuracy of the training data, the risk of overfitting and the computational effort (Mitchell, 1997; Ussath et al., 2017). Invasively, the same principle is valid for a small number of iterations and the risk of underfitting (Mitchell, 1997). Another remark of Mitchel (1997) is related to the backpropagation algorithm and its convergence over multilayer networks. The implementation of a true gradient descent method to search over the error surface for the optimal network weights, with the right learning rate, is only guaranteed to converge for a local minimum that may not correspond to the global minimum. Several approaches are listed with the objective of overcoming the problem: i) Addition of momentum term to the weight-update rule; ii) Training of multiple networks with different random weight initiations; and iii) Usage of the Stochastic Gradient Descent (SGD), instead of the standard gradient descent.

Two variations of the gradient descent methods are commonly used (Mitchell, 1997) : i) the standard, ordinary, or true gradient descent, often referred to as **batch gradient descent**, where the error is calculated over all examples (each epoch) before updating the weights (Bengio, 2012); and ii) the **stochastic gradient descent** (SGD), where error and model weights update is calculated for each example, one at a time (Bengio, 2012). While the first one provides fewer updates to the model weights and thus a more stable error calculation and convergence in the error surface, it can lead to premature convergence (local minimum), demand high for memory availability and become slow over large datasets (Bengio, 2012; Mitchell, 1997). In the other hand, the second variation, frequent weight updates can avoid premature convergence but lead to a more computational expensive performance due to a more frequent and noisier

update of the error descent path (Mitchell, 1997). A third variation of the gradient descent, also a variation of the stochastic gradient descent, called **mini-batch gradient descent**, can be used as a middle ground between the two previous techniques. By splitting the training set into small subsets (instead of only one at a time) to calculate the cumulative errors of the examples in each batch and proceeding to the update the model, the method can prove to be more adequate to avoid the limitations of the two previous methods (Bengio, 2012; Masters & Luschi, 2018).

Another optimization algorithm relevant for this work is the Limited-memory Broyden–Fletcher–Goldfarb–Shanno, commonly referred as **L-BFGS,** an optimization algorithm in the family of quasi-Newton methods often chosen for large datasets due to its fast convergence when compared with other first-order optimizations, such as the gradient descent (Apache Software Foundation, 2018g).

9.10. TRAINING / HYPERPARAMETER TUNING – ARTIFICIAL NEURAL NETWORKS

For the training and hyperparameter tuning of the Artificial Neural Networks classifier, the following values were used in order to find the average most performant set of hyperparameters over the 105 training subsets of the BEV, using a 3-fold cross validation and the AUC-PR as evaluation metric:

Table 9.7 - Neural Networks Hyper-parameter Tuning

Hyperparameter	Pyspark API	Subdivision	Values
Nodes in Hidden Layers	layers	1 hidden layer	[2], [4], [6], [8], [10], [12]
		2 hidden layers [nodes layer 1, nodes layer 2]	[4, 2] [6, 3], [6, 4], [6, 6], [8, 3] [8, 4], [8, 6] [10, 4], [10, 6], [10, 8], [10, 10] [12, 4], [12, 6], [12, 8], [12, 10] [14, 6], [14, 8], [14, 10], [14, 12], [14, 14]
Maximum Iterations	maxIter	-	10000
Solver Algorithm for Optimization	solver	-	l-bfgs, gd

The most relevant results achieved are summarized and discussed as follows:

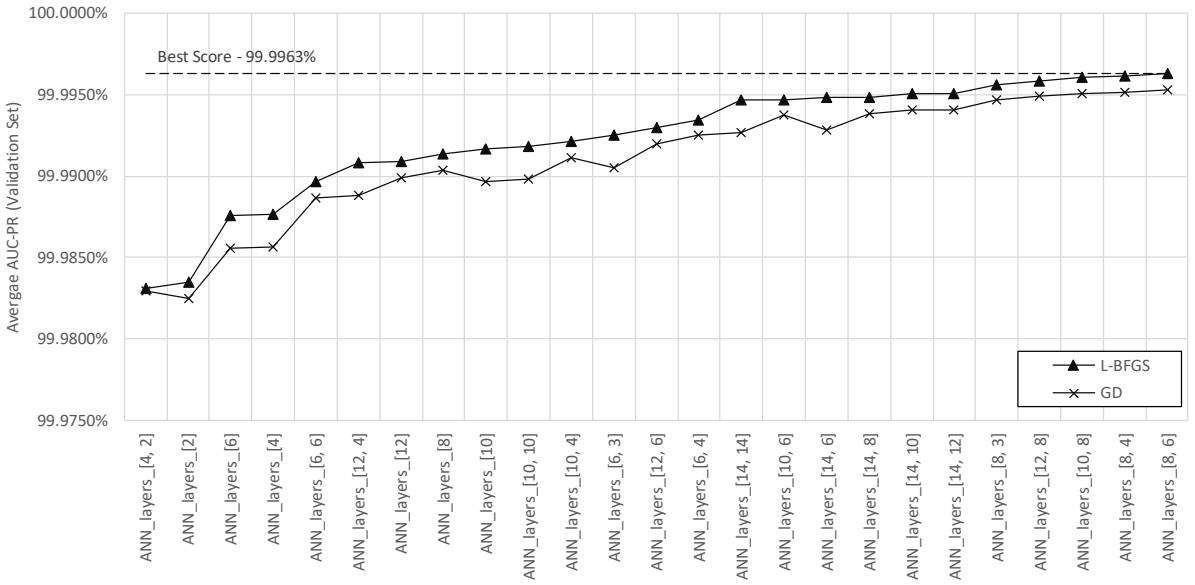


Figure 9.7 - ANN Training/Tuning AUC-PR Comparison

Overall, observing Figure 9.7, the hyperparameter combination that presented the average best scores over the validation was the network comprised by two hidden layers, the first one with eight neurons, and the second one with six neurons. In terms of scoring performance for the training/validation the registered difference between the usage of the L-BFGS or the GD optimizer algorithm is almost residual, with advantage for the first.

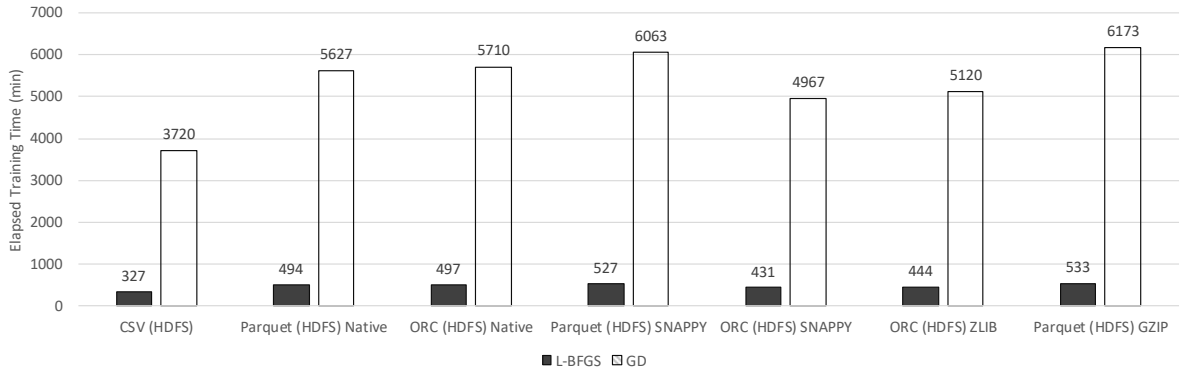


Figure 9.8 - ANN Training/Tuning Elapsed Time Comparison

However, observing Figure 9.8, the same cannot be stated about the training times, where a large gap was registered in any file format. For the best scoring performance for both L-BFGS and GD, in average the L-BFGS presented elapsed training times 91% smaller than the GD due to its fast convergence, as few iterations are required to train the classifier (not requiring to achieve the maximum number of iterations defined).

Therefore, the chosen hyperparameters to be used against the test set are as follows:

Artificial Neural Networks (ANN)	layers: 2 layers [8, 6], maxIter=10000, solver=L-BFGS

9.11. LOGISTIC REGRESSION – REGULARIZATION PARAMETERS

During the training process of the logistic regression algorithm, on each iteration, for training instances $i = 1, 2, \dots m$, the algorithm will simultaneously update all the weights θ_j values in order to minimize the average cost function ($J(\theta)$) represented by expression eq.8 (A. Ng, 2018; R. Ng, 2018):

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} log\left(h_\theta\left(x^{(i)}\right)\right) + (1 - y^{(i)}) log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right] \quad \text{(eq.8)}$$

For the previous expression, rather than highly rewarding confident and right predictions, the training will highly penalize confident but wrong prediction.

As previously mentioned, regularization parameters can be added to the cost function in order to introduce penalties capable of helping the learning process of the algorithm avoiding overfit. Using the previously presented cost function ($J(\theta)$), the regularization parameters are added as represented on expression eq.9

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} log\left(h_\theta\left(x^{(i)}\right)\right) + (1 - y^{(i)}) log\left(1 - h_\theta\left(x^{(i)}\right)\right) + \lambda\left(\alpha\left(\sum_{j=1}^{n}|\theta_j|\right) + (1-\alpha)\left(\frac{1}{2}\sum_{j=1}^{n}\theta_j^2\right)\right)\right] \text{(eq.9)}$$

Analyzing the expression, the α parameter controls the elastic net penalty distribution between the L1 and L2 norms, and the λ is the regularization parameter which controls the penalty strength. For values of α = 0, the L2 regularization will be applied. For values of α = 1, the L1 regularization is applied. For values between, the elastic net is applied with a contribution of both penalties (Apache Spark, n.d.-a).

9.12. TRAINING / HYPERPARAMETER TUNING – LOGISTIC REGRESSION

For the training and hyperparameter tuning of the Logistic Regression classifier, the following values were used in order to find the average most performant set of hyperparameters over the 105 training subsets of the BEV, using a 3-fold cross validation and the AUC-PR as evaluation metric:

Table 9.8 - Logistic Regression Hyper-parameter Tuning

Hyperparameter	Pyspark API	Values
Regularization Parameter	regParam	0.01, 0.10, 0.50
Elastic Net Penalty Distribution (L1, L2)	elasticNetParam	0.00, 0.25, 0.50, 0.99
Maximum Iterations	maxIter	10, 100

The most relevant results achieved are summarized and discussed as follows:

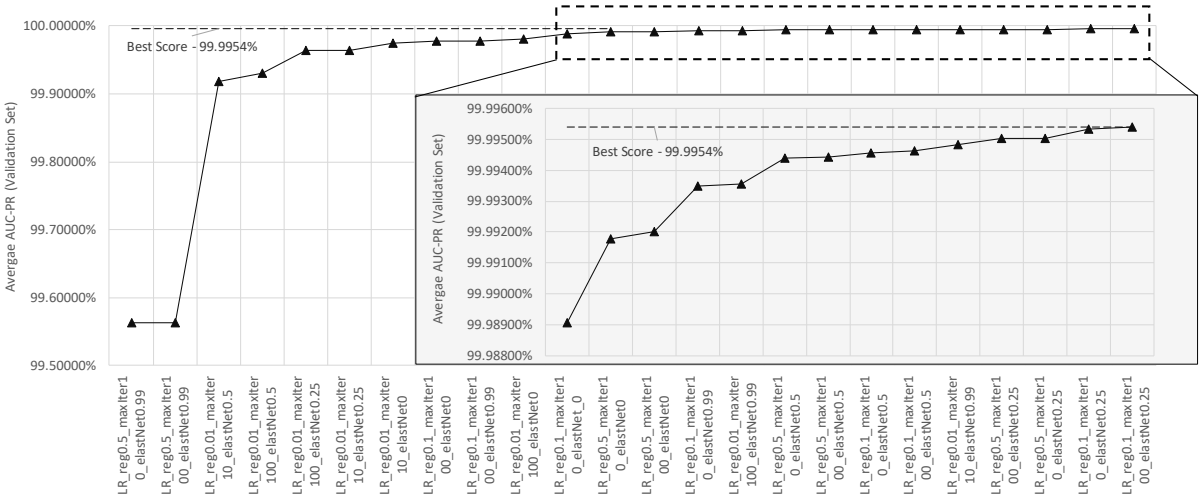


Figure 9.9 - LR Training/Tuning AUC-PR Comparison

Overall, observing Figure 9.9, the hyperparameter combination that presented the average best scores over the validation was with a regularization parameter of 0.10, with an elastic net parameter of 0.25 and a maximum number of iterations of 100.

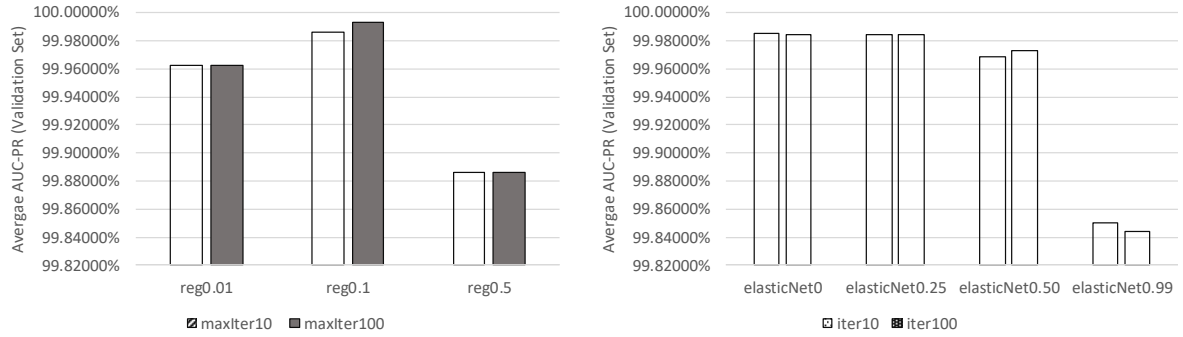


Figure 9.10 - LR Training/Validation Comparison Between Regularization and Elastic Net Parameters

Observing the previous Figure 9.10, it is possible to conclude that during training, the regularization parameter presented a validation score gain from 0.01 to 0.1, however, as the regularization parameter increases to 0.5 the score starts to decrease steeply. For the elastic net parameter, what was observed during the training was that the best scores were registered for 0 and 0.25, however, as the value increase to 0.50 and 0.99 the validation score starts to decrease. For the maximum number of iterations of 10 or 100 the validation scores produced little differences, with a small advantage for the later one.

Therefore, the chosen hyperparameters to be used against the test set are as follows:

Logistic Regression (LR)	regParam=0.10, maxIter=100, elasticNetParam=0.25

9.13. SVM - BINARY LINEAR AND NON-LINEAR APPROACHES

Among the implementations and extensions of the SVM algorithms, the following are used to address binary classification problems (Cambridge University Press, 2008a; He et al., 2016; Murphy, 2012): i) linear scheme SVM; and ii) non-linear scheme SVM.

The first one, the **linear SVM**, derives a discriminant linear function in the feature space from the training instances and their respective classes (Buczak & Guven, 2016; Murphy, 2012). Through the concept of margin, defined by the distance from the decision surface to the closest set of instances, known as support vectors, the learning algorithm is optimized through the maximization of the margin value, as can be seen on the Figure 4.15 (Buczak & Guven, 2016; He et al., 2016). The resulting approach is referred to as hard-margin SVM. This approach might, however, prove to be quite restricting and lead to a less performant generalization capacity of the classifier, especially noted if the data is not linearly separable or noisy (Murphy, 2012).

Therefore, an extension of the approach is the introduction of slack variables on the objective function representing the misclassified training instances, as expressed by the expression eq.10 (Buczak & Guven, 2016; Lardeux et al., 2009; Murphy, 2012). According to Cambridge University Press (2008) and Murphy (2012), the objective function will seek to find the optimal trade-off between the margin width and the number of points required to generate it, through the minimization of the number of the training misclassifications along with maximization of the margin, as indicated by expression eq.10, in what is referred as soft-margin SVM approach.

Given training vectors $x_i \in \mathbb{R}$, i=1,…N, in two classes, and a vector y∈{1,−1}, SVC solves the following problem:

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i \text{ subject to } \xi_i \geq 0,\ \ y_i\left(x_i^T w + b\right) \geq 1 - \xi_i\ ,\ \mathrm{i} = 1{:}\,\mathrm{N} \qquad \text{(eq.10)}$$

In other words, the training of the classifier will process through a minimization problem that will seek to find the values of w (distance between the support vector and the considered hyperplane, the margin, is equal to $1/w$, where *w* refers to the norm of *w* vector), b (bias) and ξ_i (slack variables) that maximize the margin (being max $1/\|w\|$) equivalent to min $\|w\|^2$), while minimizing the number of potentially misclassified instances penalties applied by the slack variables (ξ_i) and affected by a control parameter C that controls the number of classification errors we are willing to tolerate (Cambridge University Press, 2008c; Murphy, 2012). For instances located on or inside the correct margin boundary no penalty is applied ($\xi_i = 0$). Instances located on the wrong side of the decision boundary outside of the margin area will be considered misclassifications ($\xi_i > 1$). However, instances located inside the margin but on the correct side of the decision boundary ($0 < \xi_i \leq 1$) will not be considered misclassifications at the cost of a penalty $C\xi_i$

for the trespassing. The sum of the training errors ($\sum_{i=1}^{N} \xi_i$) can be interpreted as the upper bound of the number of misclassified instances. Finally, the regularization term C, provides a way of controlling the training overfit of the algorithm, tightening or loosening the penalties and therefore the accounting of training errors (Cambridge University Press, 2008c; Murphy, 2012).

The second one, the **non-linear SVM**, follows the same principles of the linear SVM but introduces what is referenced as the "kernel trick" in order to provide non-linear classification properties to the learning algorithm (Murphy, 2012). Through the application of a projection function (kernel function) to the training instances space, each instance can be projected and mapped to a transformed feature space. Using a non-linear kernel, the SVM derives a discriminant maximum-margin hyperplane in the transformed feature space from the training instances and their respective classes, non-linearly represented on the original input space (Boser, Guyon, & Vapnik, 1992).

9.14. TRAINING / HYPERPARAMETER TUNING – SUPPORT VECTOR MACHINES

For the training and hyperparameter tuning of the SVM classifier, the following values were used in order to find the average most performant set of hyperparameters over the 105 training subsets of the BEV, using a 3-fold cross validation and the AUC-PR as evaluation metric:

Table 9.9 - Support Vector Machines Hyper-parameter Tuning

Hyperparameter	Pyspark API	Values
Regularization Parameter	regParam	0.01, 0.10, 0.50, 1.0, 2.0
Maximum Iterations	maxIter	10, 20, 30

The most relevant results achieved are summarized and discussed as follows:

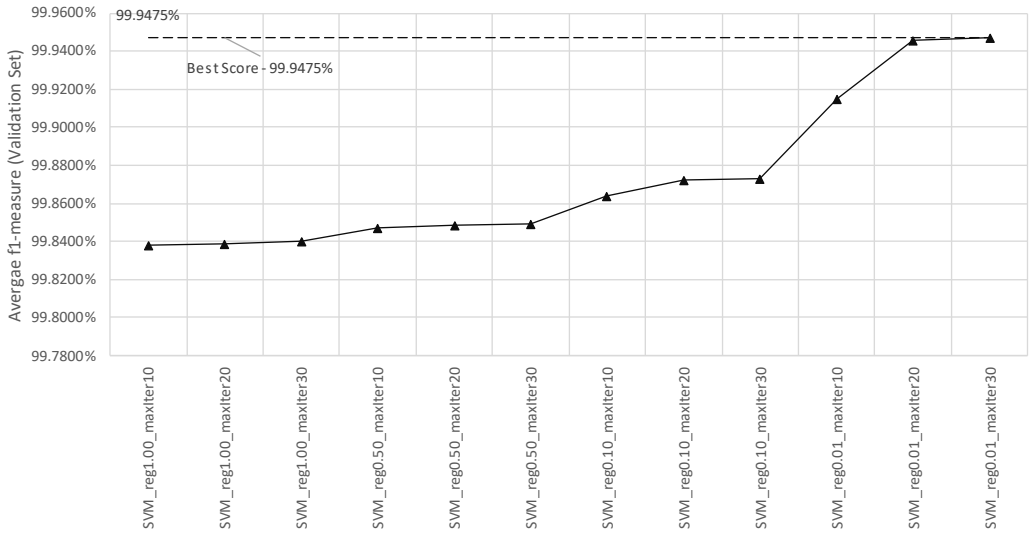


Figure 9.11 - SVM Training/Tuning AUC-PR Comparison

Overall, observing Figure 9.11, the hyperparameter combination that presented the average best scores over the validation was with a regularization parameter of 0.01, with a maximum number of iterations of 20.

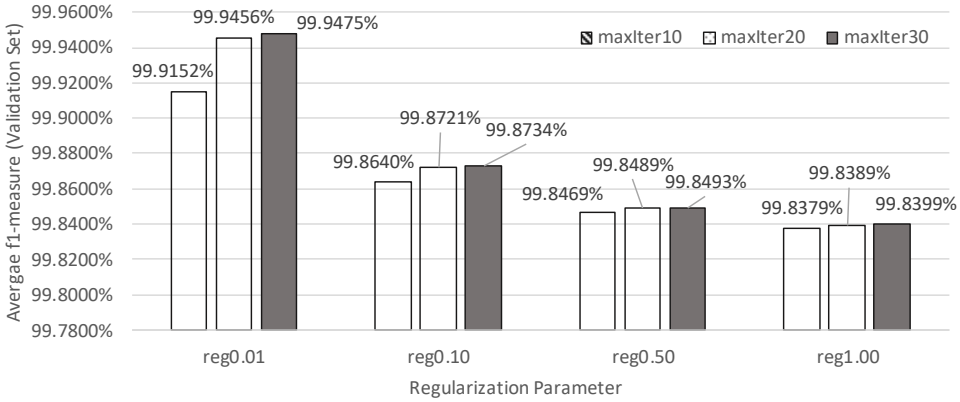


Figure 9.12 - SVM Training/Validation Comparison Between Regularization Parameters

Observing the previous Figure 9.12 where the different regularization parameters are compared, it is possible to conclude that during training using the value of 0.01 produced consistently better results than the remaining values experimented. The maximum number of iterations parameter increases from 10 to 20 where it peaks and does not produce better results onwards when trying 30 iterations. Therefore, the chosen hyperparameters to be used against the test set are as follows:

SVM	regParam=0.01, maxIter=20

9.15. RANDOM FORESTS – PSEUDO-CODE

Breiman (2001, p. 2) provides the following definition for the method: "*A random forest is a classifier consisting of a collection of tree-structured classifiers {h(x, k), k = 1, . . .} where the {k} are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input x*".

Figure 9.13 presents the pseudo-code of the algorithm:

Algorithm 1 Random Forest

Precondition: A training set $S := (x_1, y_1), \ldots, (x_n, y_n)$, features F, and number of trees in forest B.

```
function RandomForest(S, F)
    H ← ∅
    for i ∈ 1, ..., B do
        S^(i) ← A bootstrap sample from S
        h_i ← RandomizedTreeLearn(S^(i), F)
        H ← H ∪ {h_i}
    end for
    return H
end function
function RandomizedTreeLearn(S, F)
    At each node:
        f ← very small subset of F
        Split on best feature in f
    return The learned tree
end function
```

Figure 9.13 - Random Forests Pseudo-code (Bernstein, 2019)

9.16. TRAINING / HYPERPARAMETER TUNING – RANDOM FORESTS

For the training and hyperparameter tuning of the Random Forests classifier, the following values were used in order to find the average most performant set of hyperparameters over the 105 training subsets of the BEV, using a 3-fold cross validation and the AUC-PR as evaluation metric. The summarized values are as follows:

Table 9.10 - Random Forests Hyper-parameters Tuning

Hyperparameter	Pyspark API	Values
Number of Trees	numTrees	10, 20, 30, 50, 100
Impurity Measure	impurityMeasure	Gini, Entropy
Maximum Depth	maxDepth	4, 6, 8
Sub Sampling Rate	subsamplingRate	0.30, 0.60
Feature Sampling Method	featureSubsetStrategy	onethird, sqrt

Due to the different parameters involved, different visualizations were created summarizing the training and validation. The most relevant results achieved are summarized and discussed as follows:

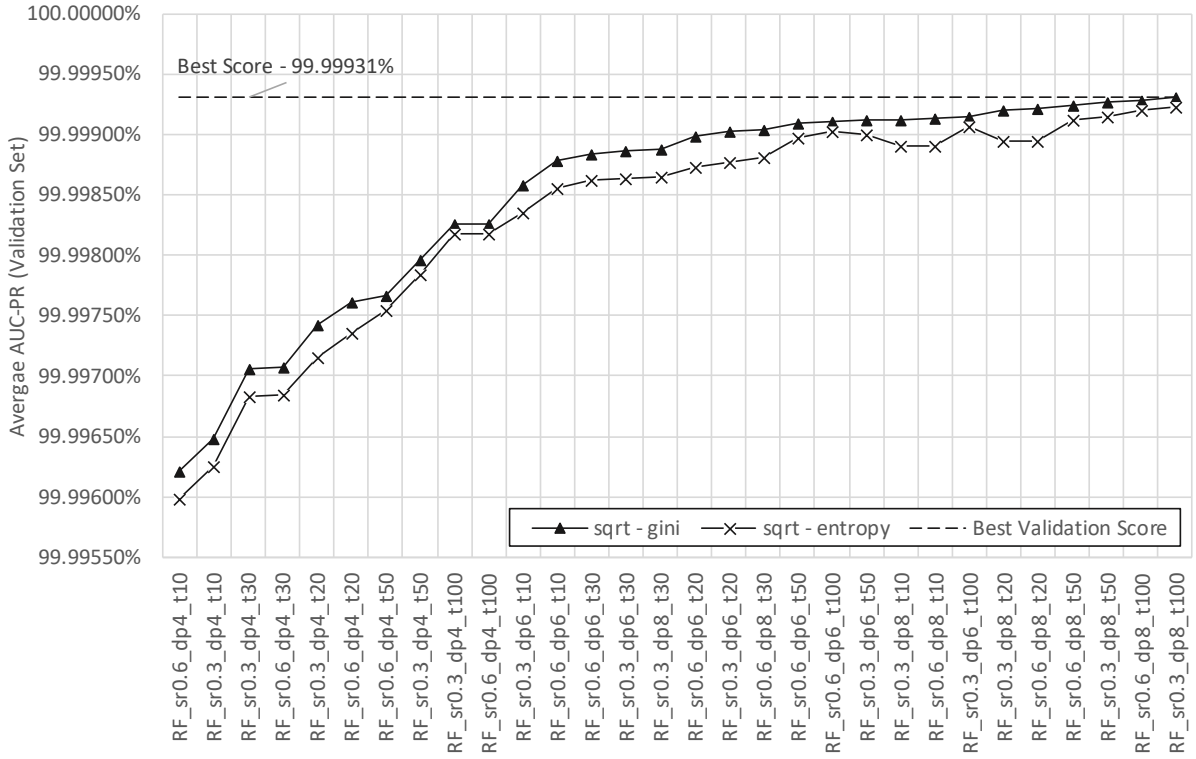


Figure 9.14 - Training/Validation Comparison Between Impurity Measures

Overall, observing Figure 9.14, the hyperparameter combination that presented the average best scores over the validation was through ensembling 100 trees, Gini as impurity measure, max depth as 8, subsampling rate at 0.30 and square root (sqrt) as feature subset strategy.

Comparing the impurity measures, the two most performant combinations resulted for the square root (sqrt) as feature subset strategy. Observing Figure 9.14, the registered validation scores presented a slight advantage for the Gini impurity measure over the entropy measure throughout the combinations assessed.

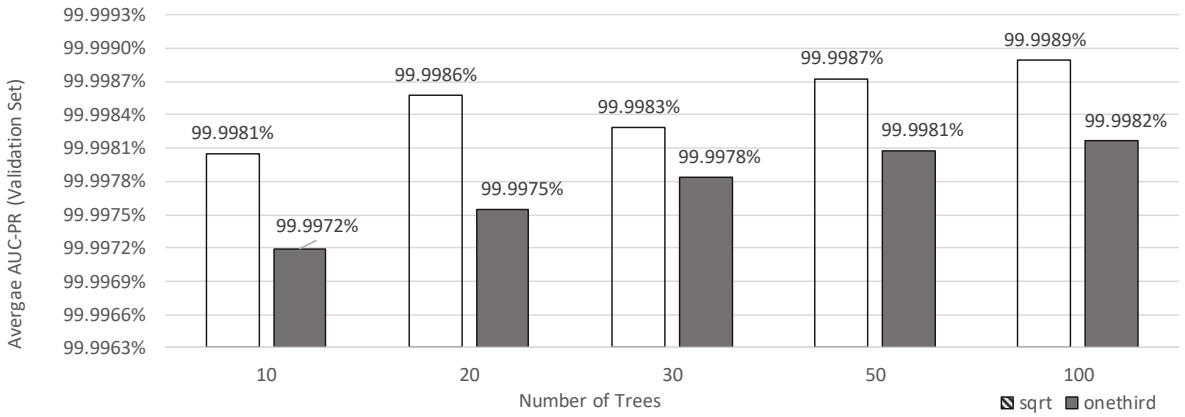


Figure 9.15 - Training/Validation Comparison Between Feature Subset Strategies

Observing the previous Figure 9.15, where feature subset strategies are compared, it is possible to conclude that during training using the "sqrt" strategy over the "onethird" produced consecutively better scoring performances as the number of trees increase, with the best scores being achieved for the 100 trees.

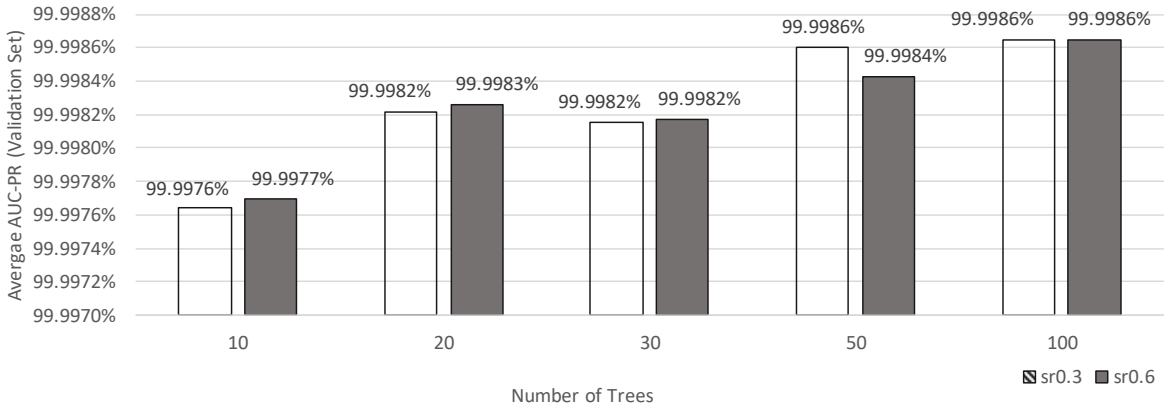


Figure 9.16 - Training/Validation Comparison Between Subsampling Rates

Observing the previous Figure 9.16, where subsampling rates are compared, it is possible to conclude that during training using the 30% strategy over the 60% produced similar results throughout the increase of the number of trees with no consistent advantage of one over the other. Ultimately, the best scores are achieved for the 100 trees scenario.

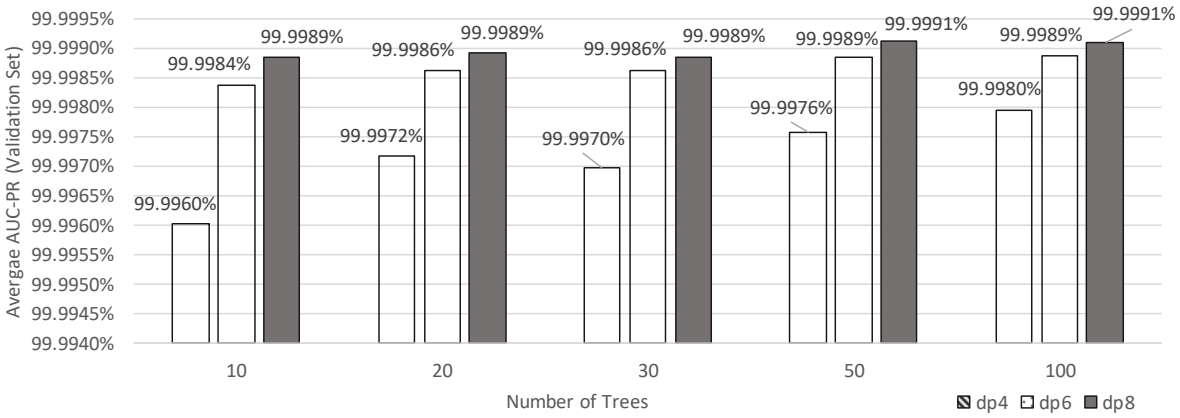


Figure 9.17 - Training/Validation Comparison Between Maximum Depth Strategies

Observing the previous Figure 9.17, where maximum depth strategies are compared, it is possible to conclude that during training using a 8 level strategy over the remaining shallower scenarios produced consecutively better scoring performances as the number of trees increase, with the best scores being achieved for the 50 and 100 trees.

Therefore, the chosen hyperparameters to be used against the test set are as follows:

Random Forests (RF)	impurityMeasure=gini, featureSubsetStrategy = sqrt, subsamplingRate=0.3, maxDepth=8, numTrees =100