DEPARTAMENTO DE INGENIERÍA
DE SISTEMAS TELEMÁTICOS
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE
TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE MADRID

DISTRIBUTED SYSTEMS FAILURE MANAGEMENT
THROUGH APPLIED MACHINE LEARNING

Autor: JOSÉ MANUEL NAVARRO GONZÁLEZ

Ingeniero de Telecomunicación

Director: JUAN CARLOS DUEÑAS LÓPEZ

Doctor Ingeniero de Telecomunicación

2018

Tribunal nombrado por el Magfco. y Excmo. Sr. Rector de la Universidad Politécnica de Madrid, el día ___ de _____ de 2018.

Presidenta: _____

Vocal: _____

Vocal: _____

Vocal: _____

Secretario: _____

Suplente: _____

Suplente: _____

Realizado el acto de defensa y lectura de la Tesis el día ___ de junio de 2018 en la E.T.S.I.T. habiendo obtenido la calificación de _____

LA PRESIDENTA

LOS VOCALES

EL SECRETARIO

Distributed Systems Failure Management Through Applied Machine Learning

A DISSERTATION PRESENTED

BY

JOSÉ MANUEL NAVARRO GONZÁLEZ

TO

THE DEPARTAMENTO DE INGENIERÍA DE SISTEMAS TELEMÁTICOS

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTORADO EN INGENIERÍA DE SISTEMAS TELEMÁTICOS

IN THE SUBJECT OF

AUTONOMOUS SYSTEM MANAGEMENT

UNIVERSIDAD POLITÉCNICA DE MADRID

MADRID, SPAIN

JUNE 2018

© 2018 - *JOSÉ MANUEL NAVARRO GONZÁLEZ*
ALL RIGHTS RESERVED.

Distributed Systems Failure Management Through Applied Machine Learning

ABSTRACT

This thesis deals with the problem of managing failures on distributed systems, specially on computer networks and high performance computing clusters. Through it, I expose and analyze the importance of the problem and how its current research landscape, while extensive, is fragmented, isolated and takes a too narrow approach. Specially, there is a gap of knowledge between academic and industrial problems and the need for a human expert and all of the problems that this entails have been overlooked. Based on this situation, I take two real datasets, a public one, detailing errors occurred on a supercomputer at Los Alamos, USA, and the other obtained from a Spanish bank's computer network, containing events extracted from a commercial network manager. With them, I propose four different contributions: a detailed study of an array of Machine Learning models, a novel optimization method to decide which time period to observe in the past to perform future prediction, a way to extract potential error causes from the prediction models and a practical implementation of these concepts utilizing Big Data software. Results show that my proposals are able to achieve successful solutions with minimal human interaction needed and satisfying technical requirements and limits.

Esta tesis trata el problema de la gestión de los fallos en sistemas distribuidos, especialmente en redes de ordenadores y clusters de computación de alto rendimiento. En ella, expongo y analizo la importancia de éste problema y cómo las investigaciones actuales son extensas pero fragmentadas y aisladas, con un enfoque demasiado estrecho. Especialmente, hay un vacío de conocimiento entre los problemas académicos y los problemas industriales. Además, la necesidad de un experto humano y todas las tareas que esto conlleva es algo que no se ha tratado en profundidad. Partiendo de esta situación, tomo dos conjuntos de datos reales: uno público, que contiene los errores ocurridos en un supercomputador en Los Álamos, EE. UU., y el otro obtenido de una red de ordenadores de un banco español, que detalla eventos extraídos de un gestor comercial de red. Con ellos, propongo cuatro contribuciones diferentes: un estudio detallado de un conjunto de algoritmos de Aprendizaje Automático, un método novedoso de optimización que permite decidir qué período temporal observar para realizar predicción de fallos, una manera de extraer causas de errores potenciales de los mismos modelos de predicción y una implementación práctica de estos conceptos utilizando software Big Data. Los resultados muestran que mis propuestas son capaces de conseguir soluciones exitosas con una interacción humana mínima, además de satisfacer los requerimientos y limitaciones técnicas.

Contents

1	INTRODUCTION	1
1.1	Methodology	5
2	STATE OF THE ART	23
2.1	Failure Prediction	24
2.2	Open Problems	28
2.3	Proposed Solutions and Their Foundations	35
2.4	Chapter Summary	55
3	OBJECTIVES	56
3.1	Solution Requirements	57
4	DATASETS DESCRIPTION	60
4.1	Computer Network Dataset	61
4.2	LANL Dataset	76
4.3	Data Preprocessing	88
5	CONTRIBUTION 1: MACHINE LEARNING MODELS COMPARISON	90
5.1	Experiments Description	91
5.2	Model Results	93
5.3	Performance Results	104
5.4	Computation Time	105
5.5	Contribution Summary	105

6	CONTRIBUTION 2: OPTIMIZING OBSERVATION WINDOWS	107
6.1	Problem Definition	107
6.2	Proposal Definition	109
6.3	Validation	116
6.4	Experiments Results	118
6.5	Chapter Summary	128
7	CONTRIBUTION 3: A METHOD FOR MACHINE LEARNING BASED ROOT CAUSE ANALYSIS	131
7.1	Proposal Definition	131
7.2	Validation Experiments	133
7.3	Analysis of system-wide properties	135
7.4	Identification of Causes of Errors	141
7.5	Miscellaneous Analysis	143
7.6	Contribution Summary	147
8	CONTRIBUTION 4: ONLINE FAILURE PREDICTION IMPLEMENTATION ON A BIG DATA PLATFORM	149
8.1	Design Proposal	150
9	CONCLUSIONS	162
9.1	Methodological Risks and Limitations	165
9.2	Future Work	167
	REFERENCES	180

Listing of figures

1.1.1 Basic components of the Design-Science Paradigm.	7
1.1.2 Basic research activities of the Design-Science Paradigm.	8
1.1.3 Disciplines that compose the Data Science process [1].	11
1.1.4 Steps of a Data Science project.	13
1.1.5 Actual methodology steps I followed on my research.	21
2.1.1 Basic Online Failure Prediction concepts.	24
4.1.1 Event distribution of the Computer Network dataset.	62
4.1.2 Event distribution of the Computer Network dataset without the outlier.	63
4.1.3 Monthly event count of the Computer Network dataset.	64
4.1.4 Event distribution per weekday on the Computer Network dataset.	65
4.1.5 Event distribution on the Computer Network dataset divided by Severity.	66
4.1.6 Event distribution on the Computer Network dataset divided by Severity and weekday.	66
4.1.7 Event distribution on the Computer Network dataset divided by Severity and weekday without Blank events.	67
4.1.8 Event distribution of the Computer Network dataset by device type and Severity.	68
4.1.9 Event distribution of the Computer Network dataset by device.	69

4.1.10	Event distribution of the Computer Network dataset by device.	69
4.1.11	Critical Event 1 monthly distribution.	71
4.1.12	Critical Event 1 distribution in December.	71
4.1.13	Critical Event 1 monthly distribution separated by device.	72
4.1.14	Critical Event 2 monthly distribution.	72
4.1.15	Critical Event 2 monthly distribution separated by device type.	73
4.1.16	Critical Event 3 monthly distribution.	74
4.1.17	Critical Event 3 monthly distribution separated by device.	74
4.1.18	Critical Event 4 monthly distribution.	75
4.1.19	Critical Event 4 monthly distribution separated by device.	75
4.2.1	Event distribution of the LANL dataset.	78
4.2.2	Event distribution of the LANL dataset separated by error type.	78
4.2.3	Event distribution of the LANL dataset separated by system.	79
4.2.4	Ratio of errors per number of nodes for each system of the LANL dataset.	80
4.2.5	Time amount each system of the LANL dataset was deployed, measured in days.	81
4.2.6	Normalized ratio of errors per number of nodes and time deployed for each system of the LANL dataset.	81
4.2.7	Normalized ratio of errors per time deployed for each node of the LANL dataset.	82
4.2.8	Number of hardware events over time divided per system on the LANL dataset.	85
4.2.9	Number of software events over time divided per system on the LANL dataset.	86
4.2.10	Number of undetermined events over time divided per system on the LANL dataset.	87
4.2.11	Number of undetermined events over time divided per system and type of event on the LANL dataset.	87
5.2.1	Kappa for each tested event based on the number of trees grown.	94

5.2.2	Kappa for each tested event based on the number of trees grown and the split rule.	95
5.2.3	Kappa for each tested event based on the the minimum node size.	96
5.2.4	Kappa for each tested event in terms of the complexity parameter.	97
5.2.5	Kappa for each tested event in terms of the cost parameter.	98
5.2.6	Standardized Kappa of the SVM models in terms of their parameters. Green color indicating that the model obtained a Kappa higher than 2 times the standard deviation of the set.	99
5.2.7	Kappa for each tested event in terms of alpha on the Elastic Net LR models.	101
5.2.8	Kappa for each tested event in terms of lambda on the Elastic Net LR models.	101
5.2.9	Kappa for each tested event in terms of alpha taking only values such that $0 < \lambda \leq 0.15$	102
5.2.10	Kappa for each tested event in terms of the number of boosting iterations on the Boosted Logistic Regression models.	103
6.2.1	Steps of my custom Genetic algorithm. Grayed out boxes indicate customized steps.	113
6.4.1	Genetic algorithm behavior for event S1.	119
6.4.2	Genetic algorithm behavior for event N2.	121
6.4.3	Genetic algorithm stability for 10 runs of different values of a in terms of median AUC, median fitness and median window length of the individuals of the last generation.	123
6.4.4	Window configurations for the individuals of the last generation of 10 different Genetic Algorithm runs for events S1 (left) and N4 (right).	124
6.4.7	Windows selected for event N4 by the Genetic Algorithm and its closest heuristic counterpart.	128

6.4.5	AUC vs window length results for the Genetic Algorithm versus the heuristic approach for the High Performance Computing dataset.	129
6.4.6	AUC vs window length results for the Genetic Algorithm versus the heuristic approach for the Networking dataset.	130
7.1.1	Proposed RCA methodology.	133
7.2.1	AUC distribution of the trained models.	135
7.3.1	Influence Matrix depicted as a heatmap.	136
7.3.2	Influence graph of the top 1 percent influences, including communities.	139
7.3.3	Summarized devices graph.	141
7.4.1	CPU threshold violation critical event chain influences.	143
7.5.1	Danger Index distribution for the events on the system for $\alpha \in \{0.9, 0.95, 0.99\}$	145
7.5.2	Event distribution based on their Mean Influence Index and Number of Affected Events for Major and Critical events.	147
8.1.1	Online Failure Prediction System Architecture Proposal.	152
8.1.2	Training Subsystem and Prediction Subsystem execution examples.	154
8.1.3	Average F1 score for the models based on tree depth, feature selection strategy and splitting criterion.	156
8.1.4	Average F1 Score and training time based on tree depth for the two best feature selection strategies.	156
8.1.5	Average F1 Score obtained for each model. Color indicates the model severity: Blank (gray), Minor (green), Yellow (major) and Critical (red).	157
8.1.6	Visualization tool example.	159

A VICENTE NAVARRO HIDALGO.

Acknowledgments

I can't start this section in any other way than recognizing the massive debt I owe to Juan Carlos Dueñas, my thesis advisor, boss and friend for the last five years. My life has changed in more ways that I could reflect here and I've learned and grown along this years. And I owe it all to you. A deep, sincere, thanks. And I also could not continue without thanking my workmates (and friends): Álvaro, Antonio, Paco, Bea, Naye, Hugo, Mario, Pablo, Samuel... But, even more specially, Javier Andión, who has suffered me singly on the last period of our lab. For that, for every hour we spent in front of the whiteboard designing crazy systems, for every stupid question I asked about Big Data and algorithms and, definitely, for being the best programmer I know, I thank you. I couldn't have done this without you.

Getting aside from the academic side of things, the person who has suffered me on my long nights of coding and writing has been Marina. You saw me finish my first degree, start my Ph.D. and are going to see it finished. You've made this marathon a walk in the park. I could not ask for more from anyone. Thank you. And while they haven't had to suffer me coding and rambling, the foundations of who I am were given to me by my parents and my sister. My success is bonded to yours.

Lastly, to all of the friends I've made and the people I've met, including my Japanese friends: thank you for walking with me.

To everyone I've named and to those I haven't: you've made these five years the happiest period of my life. Thank you.

No puedo empezar esta sección de ninguna otra manera que reconociendo la increíble deuda que tengo con Juan Carlos Dueñas, mi director de tesis, jefe y amigo durante los últimos cinco años. Mi vida ha cambiado en más sentidos de los que puedo reflejar aquí y he aprendido y crecido con los años. Y te lo debo todo a ti. De corazón, gracias. Tampoco podría continuar sin recordar a mis compañeros (y amigos): Álvaro, Antonio, Paco, Bea, Naye, Hugo, Mario, Pablo, Samuel... Pero, mas aún, a Javier Andión, que me ha sufrido en solitario en el último período de mi doctorado. Por eso, por cada hora que pasamos delante de la pizarra diseñando locuras, por cada pregunta estúpida que te hice acerca de Big Data y algoritmia y, sin duda, por ser el mejor programador que conozco, gracias. No podría haber hecho esto sin ti.

Dejando de lado el aspecto académico, la persona que me ha sufrido durante las largas noches de escritura y programación ha sido Marina. Me has visto terminar una Ingeniería, empezar el doctorado y me vas a ver terminarlo. Has hecho que esta maratón sea un paseo en el parque. No podría pedir más de nadie. Gracias. Y aunque ellos no me han sufrido programando y quejándome, los cimientos de quien soy han sido formados por mis padres y mi hermana. Este éxito está cimentado en los vuestros.

Por último, a todos los amigos que he hecho y la gente que he conocido, incluyendo mis amigos japoneses: gracias por andar conmigo.

A todos los que he nombrado y a los que no lo he hecho: habéis hecho estos cinco años el período más feliz de mi vida. Gracias.

1

Introduction

There are several areas to take into account when managing a computer network or a system. FCAPS [2], which stands for fault, configuration, accounting, performance and security, is the ISO standard for network management and gives a good overview of certain aspects to consider for this task. While originally designed just for networks, certain parts of it can be extrapolated to distributed systems in general. Indeed, its first part, faults (understood as any event that has a negative significance for the system), is a pervasive threat to all of them: no system is immune to faults, whether they are hardware or software-based. And this can be a huge problem, as distributed systems are a crucial part of our technologies. In fact, to understand how important the presence of distributed systems is, laying down some definitions is needed. Tanenbaum [3] defines a distributed system as:

”A distributed system is a collection of independent computers that appears to its users as a single coherent system.”

This definition sounds simple enough. But it entails a lot behind it. First of all, appearing as a single system is something easier said than done. Some of the ways in which this transparency must be achieved are: location (the actual physical location of the system must be hidden to the end user), migration (parts of the system should be able to be changed without the user noticing it), replication (the system must be able to have redundancy without it affecting the end user), concurrency (the end user should be oblivious to the background computations and their coordination) and failure (the system must hide the occurrence of failures and be able to keep on functioning with minimal disturbance to the user). Ensuring these (and many more) is a daunting task, but a crucial one: the downtime costs of a data center can get up to half a million dollars per hour in some cases [4].

The definition implies another fact, though. If we take a moment to study it we'll find that they are almost omnipresent in our current lives. Indeed, some obvious examples may come to our minds: computer networks, sensor networks, Internet of Things devices, high performance computing clusters, Big Data analysis clusters. But this only scratches the surface of their importance. It is made clearer when we start thinking about some of the applications of the systems we've named: computer networks can range from the small network of a company to Amazon S3 or Cloudflare, the services that host thousands of the most visited websites on the planet. Sensor networks can be a small mesh of thermometers on a Smart Home or the infrastructure that monitors the safe functioning of a nuclear plant. Thus, they have become so important that a fault (or failure, as I'll refer to them from now on) on them can lead to situations that range from minor nuisances (a website being offline for some time) to life-threatening (the sensor network of the nuclear plant shutting down or malfunctioning). While twenty years ago, ensuring that a network stayed up was a business requirement, today a network shutting down may mean the disappearance of a significant percentage of the most popular websites. This change in the importance of these systems demands an analogous change on the way they are managed. The classical approach to managing failures is a reactive one: detecting their appearance, fixing them and recovering the system as soon as possible. This is good enough for most systems, but when they must be online no

matter what happens, this ceases to be enough. Future management techniques must be proactive: in order to avoid any system downfall, they must be able to predict when and where a failure will happen and minimize its harm, if completely fixing it beforehand is not possible.

Modern systems also present other challenges that disrupt classical failure management (and I'm going to utilize some of Big Data's V's as a template for this, as it serves as an appropriate analogous):

- Volume: as business and companies grow, so do the systems where they rest. Current networks and distributed storage systems work with potentially hundreds or thousands of devices, which must be managed at the same time, in runtime.
- Velocity: a consequence of the volume and the need to analyze all the monitored system as it runs, the processing power to analyze everything at the same time increases exponentially.
- Variability: another consequence of distributed systems becoming widespread is their specialization, each system will be tailored to perform certain tasks with certain specific devices. This makes knowledge generalization very hard, which in turn creates isles of knowledge, where how to do things for a certain kind of system may not be appropriate for other, similar ones.

These conditions obligate us to begin drifting away from human system managers. A human will not be able to keep up with the amount of information and devices that need to be processed and analyzed. Furthermore, a manager will need to be specially trained for their specific kind of network or distributed system, a process that is costly, both in money and time, as well as risky, as that knowledge is gone if the manager is. If we join this conditions together with the need for proactivity, it is clear that future distributed system managers must be autonomic. And this is, in fact, the path that has been taken for some years now: there are innumerable computer network manager software, for example, that monitor each device in real time and create summaries and metrics to help the operators. Only on recent

years we are starting to also see some efforts in bringing a semblance of autonomy to these products, with some pattern seeking and anomaly detection capabilities. But those are far from complete solutions. And they are, normally, commercial products, as well, so their theoretical foundations cannot always be checked. I cannot say, either, that the scientific literature side of this issue has presented a complete solution. As we'll see on the next chapter, the issue of Online Failure Prediction, specially in networks, presents a fragmented landscape. While there is an extensive body of work on it and a myriad of different Machine Learning algorithms and solutions have been tested, there are certain core open problems that have not been addressed and hinder the possibilities of reaching better solutions. These are:

- Lack of public benchmark datasets: this is only a present problem on the area of Network Event Failure Prediction, but a great one. As there are no commonly used datasets (such as MINST for digit recognition, for example), most solutions cannot be compared between each other in terms of predictive quality, as what may be very good for a certain dataset may not be so for another one.
- Ad-hoc solutions: most of the works I've read on this area seem to test a specific idea or algorithm on their specific dataset. While that is still valuable, it is a limited approach. We should strive to find general solutions that require no human interaction or expertise in order to fulfill the constraints I previously talked about.

What is more, while studying the state of the art of Online Failure Prediction, I discovered that some situations that arise naturally when you are working on a realistic environment have not been treated at all on scientific literature. Namely, how to select which time period to use as evidence to predict future events (which normally requires system-specific knowledge), how to obtain probable failure causes without human expertise needed and how to deploy an Online Failure Prediction solution on a real system. These are all real problems that need to be tackled. And

I do so on this thesis, using two real datasets from two different industrial environments to help validating each proposal. This thesis has four relevant research contributions. First, I will compare the performance of a large number of different Machine Learning algorithms, in order to find the best performing ones for my datasets. And then, I present solutions for the three realistic problems I've talked about on this paragraph. For the time period selection issue, I propose the usage of a customized genetic algorithm alongside a complex multi-window scheme that requires no previous expertise about the modeled system and optimizes both model performance and storage costs. To reduce the need of a human expert when extracting potential failure causes, I propose the creation of an Influence Matrix, a highly summarized representation of every model that, nevertheless, contains enough information to extract all kinds of insights and information about the system they model. And, lastly, for the deployment problem, I propose a two-system architecture based on Big Data software components that are the current industrial standard for their purposes. While on the Methodology subsection I'll explain in further detail the requirements this solutions presented, a key factor that has guided all the steps I've taken on this thesis has been an effort to abstract human knowledge and interaction with the systems. As such, all of my contributions need minimal human input and don't need any expertise on the modeled system. As I said, this is one of the key features that future Failure Management systems will have. As a sidenote, while Security problems are close to the area I will talk about and can, indeed, cause failures, they are treated using a different technique than Online Failure Prediction called Anomaly Detection. As such, Security problems will not be treated in this work.

1.1 METHODOLOGY

I've already mentioned that my thesis is composed of four different contributions, three of which are solutions to problems that only arise when the Failure Prediction task is taken to realistic, practical environments. This closeness to industrial solutions has influenced in the methods and approaches I've taken when carrying

out my research. Specifically, I've been influenced by two different methodologies. First, I'll define both of them, and then I'll detail how I've used them for my research.

1.1.1 DESIGN-SCIENCE PARADIGM

This methodology, explained profusely in [5], lays out a paradigm that seeks to standardize how research in technical areas can be carried out in business environments. As my research has been developed closely to industrial and realistic ones, this proposal is very adequate to my situation. Another reason why I adopted Design-Science is because it helps dealing with "wicked" problems, ones in which the requirements are unstable, the problems have complex interactions and there is a great dependence on human cognitive abilities or in human teamwork. We'll see that most of the contributions my thesis addresses present several of these features. In fact, one could say that most of my work has been done in order to reduce the need for human expertise or action in the Failure Management process as much as possible. Going back to the methodology itself, one of the main ideas this paradigm proposes is the concept of "artifacts", a series of concepts that allow researchers to understand and address a problem. They are more easily understood as the building blocks of research projects, and they are divided into four different categories:

- Constructs: vocabulary and symbols, the specific language in which problems and solutions are defined.
- Models: abstractions and representations of the specific problem and its solution space.
- Methods: the algorithms and practices that guide the search for a solution on the solution space.
- Instantiations: the implementations and prototypes that show how the defined methods, models and constructs can be brought to reality. Ultimately, these are the ones that show the feasibility of a research project.

As such, an artifact can take many shapes: software, formal logic, mathematical proofs, specific algorithms or even just natural language ideas. The original framework that the Design-Science Paradigm proposes is based on three components (Fig. 1.1.1): the environment, or business needs, defines the problem to solve and the solution space, the range of possible solutions to it. It goes without saying that, in my situation, instead of business needs, I worked based on open problems on the Failure Management area. After receiving the problem to address, the researchers, utilizing the methods and previous works found on the knowledge base, try to solve the business needs through the creation of one or several artifacts. If successful, this solution is then incorporated to the knowledge base and will remain, from then, as a part of it for future researchers to consult.

Figure 1.1.1: Basic components of the Design-Science Paradigm.

Of course, apart from the definition of artifacts, the crucial part to define is how researchers should solve a problem. This paradigm defines two complementary phases to do so: Building and Evaluation. These concepts are summarized on Fig. 1.1.2: researchers engage on a Build-Evaluate loop, where a solution, an artifact, is created and evaluated following well-known methods. This evaluation provides feedback and illuminates the problem, this, in turn, shapes how to improve the solution until the problem at hand is solved in a way that satisfies every requirement. This paradigm also takes into account the fact that, sometimes (as we'll see happens, in fact, for some of my contributions) there is no previous solution or method to improve. On these cases, the only option is a combination of creativity and trial and error searches.

Figure 1.1.2: Basic research activities of the Design-Science Paradigm.

Lastly, Design-Science gives a series of guidelines that inform the features a well-built artifact should have, as well as the process a researcher should follow. These are:

1. Design an artifact: the final objective of the research process should always be to obtain an artifact that solves the addressed problem. As such, it must be defined extensively and precisely, allowing for it to be implemented in the appropriate domain.
2. Design the artifact for a relevant problem: the problem that the artifact deals with must be important and, either, unsolved or done not efficiently enough.
3. Design the evaluation: the solution is only as good as it has been shown to be. How, exactly, this evaluation is done, must be informed by the business needs and shaped by the known methods. One relevant part of this evaluation is how to incorporate the solution to the environment it was developed for.
4. Clearly define the contributions: ultimately, some part of the solution must be novel. It could be the artifact itself, new constructs, models, methods or instantiations developed on the design process or the proposal of new evaluation methods.
5. Maintain research rigor: this is derived from basing the theoretical foundations and applied methodologies on the current knowledge base.

6. Design the solution using a search-based process: the business needs define both the problem and the possible set of solutions, the solutions space. Thus, the process of designing the artifact must be a process of searching and testing between the different available solutions in order to find the optimal one.
7. Disseminate your results: again, your solution is only as good as you can show it to be. An artifact that is never shown to anyone is the same as a non-existent artifact.

While these guidelines were given thinking about scientists inside businesses, they can be applied almost directly to a purely scientific situation. For example, guideline 7, about communicating results, is still applicable as-is to university-based scientists: a perfect research that is never published is useless.

Overall, this Paradigm has shaped profusely how my research has been carried out. I will now define how another kind of methodology has influenced my works and, after that, I'll lay out the specific steps I followed, combined from both proposals.

1.1.2 DATA SCIENCE

Data science is an interdisciplinary field of work and research whose purpose is to extract, from a limited amount of data, generalizable knowledge that will allow to make useful predictions. To be able to do so, it encompasses all the distinct phases which information must go through to be transformed into data, cleaned for their analysis, the construction of models that best explain them and their implementation on a production system. Because of this, data science comprises and uses a lot of disciplines, such as statistics, probability analysis, mathematics, data warehousing, coding, signal processing, data visualization and data engineering. Furthermore, recent advances in the “datification” of everything have made the global amount of available data grow exponentially, so, nowadays, a data scientist has to be able to work with big data platforms, too. Due to the amount of knowledge this

kind of work requires, there are few data scientists officially educated on it. Most of the data scientists' body is composed of statisticians, engineers and computer scientists. This amalgam of requirements is also one of the reasons why data science has not been completely defined until recently: statisticians didn't know how to code profusely and computer scientists didn't have enough statistics background (related to this, one informal definition of a data scientist is "a person who knows how to code better than a statistician and knows statistics better than a computer scientist").

Historically, as I've already said before, data science is a relatively new field. The first mentions of something related to it were on 1962, by John Tukey, on [6], when he wrote how he had always believed to be a statistician but now he viewed himself as a data analyst. Apart from that, Tukey himself established one of the main steps of a data science project (which we'll see on next) on his book, *Exploratory data analysis* [7]. 1974 was the year when the term data science appeared clearly and with the meaning we give it today, by Peter Naur, on his book, *Concise Survey of Computer Methods* [8], a term which he had already used previously, together with "datology", as a synonym for computer science, whereas in this book he defines it as "The science of dealing with data, once they have been established, while the relation of the data to what they represent is delegated to other fields and sciences." Probably the most important step towards the establishment of data science as a serious research field was the creation, in 1989, of the *Knowledge Discovery in Databases Workshop*, by Gregory Piatetsky-Shapiro, which is still the main workshop of this area, and the *Data Mining and Knowledge Discovery Journal*, in 1997.

The reader will probably be familiar with the terms data mining, machine learning and statistics, and will be probably wondering what's the difference between a data scientist and a statistician or how data science differs from applying machine learning techniques to a dataset. The best analogy would be to think of data science as a "supergroup" in music (a music group formed by artists who are already notable or respected in their fields): data science uses statistics, machine learning and data mining techniques as tools to solve specific problems. It is a step above of all

those. According to Drew Conway [1], data science can be said to be standing over three main foundations (Fig. 1.1.3: hacking skills, statistics and maths knowledge and, the differentiating factor for data science from everything else we've talked about before, environment expertise and a scientific approach to problems. Let's analyze each factor.

Figure 1.1.3: Disciplines that compose the Data Science process [1].

- First of all, why hacking skills and not computer science? Because theoretical computer science knowledge is not an absolute requirement for the kind of work needed with data on a computer. Even though it, of course, helps, the required knowledge can be acquired outside of a computer science degree.
- On the upper side of the diagram we can see where statistics and machine learning lie and the main difference between a statistician and a data scientist.
- Lastly, it's important to highlight the danger zone: people who know how

to analyze data and have the scientific view of proposing hypothesis and know how to test them, but don't have enough background in statistics. This can result in the application of wrong techniques or bogus results, which permeates the false idea that statistics lie.

Once we have seen the differences between data science and other a priori similar disciplines, we can start to discern its usual purpose: data science is an applied mixture of disciplines that is used to solve real problems, from daily ones to complex, technological ones. As such, several procedures, conventions and work protocols have been de facto adopted by most data scientists to tackle these problems. We'll study them next.

DATA SCIENCE PROJECT STEPS

Based on previous works [9] and my own experience on data science projects, there are several different jobs to complete when dealing with a project of such kind. But, before getting to them, there are several terms regarding the possible roles in the project that need to be defined:

- Data Science team: the team that will develop the solution to the problem posed by the project sponsor. It will generally be composed of several people, each with a strong background in each discipline needed for the project, including statistics, computer science and (this is usually forgotten) communication skills. From now on, it will just be referred as the team.
- Project Sponsor: this is the most important person in a data science project. By definition, the project will be a success if and only if the project sponsor approves it. Generally, it represents the business interests.
- Project Client: a key part of the project. If the sponsor represents the business interests, the client is the one that the data science solution will affect directly. As such, it serves as a domain expert and also as a reference to check the real utility of the solution on their daily work.

- **Data Architect:** the one in charge of managing all the data. There are two dimensions to this role: first, the data must be extracted from the sponsor's/client's organization and supplied to the team and then, the team's architect must manage those data to best serve the purpose of the team. It makes sense, then, to assume that usually there are two or more data architects for each project.
- **Operations:** the person/team in charge of deploying the data science solution on the organization. They will usually impose constraints about the size of the software to deploy, its programming language, the data it can access to and its response time.

Having defined the different possible roles, let's examine the usual steps in which a project is usually divided into (shown in Fig.1.1.4):

Figure 1.1.4: Steps of a Data Science project.

Problem and goal definition: even though it may seem obvious to start here, there's more than meets the eye at this step. The problem definition, as with most software projects' requirements definition, will affect profoundly how the data science team will work in the next parts. It is on this part where one of the main features of this discipline can be seen more clearly: the exchange of communication between the working team, the project sponsor and the client. It is at this point, too, where final constraints imposed by the operations team should be taken into account, as they can influence several decisions to be taken. There are two possible kinds of problems:

- Closed problems: in this kind of projects, the problem and its possible solution is clearly defined. Usually, the sponsor knows what data are related to the problem. These are the simpler ones (though that doesn't mean they're easier: the causal assumptions could be wrong or there could be more data not detected that influences the problem, which would cripple the analysis).
- Open problems: whether it is because the cause is not known, because there is no exact problem to be studied or because there is a big amount of data possibly related to the problem, these kind of problems offer more freedom to the team. The definition could go from just exploring data to improve whatever the team detects to finding the cause of an unknown problem.

Another key point to remark is that it is on this phase where the translation of the sponsor's requirements to scientific terms should happen, which settles the exact objectives that the solutions must meet. We'll see more about the usual metrics used in further sections.

On this phase, there are several key decisions that must be taken: which data will be gathered, the exact situation that the sponsor wants to solve/improve, what is the condition that will make the project successful and the deadline for the project.

Data gathering: after speaking with the client and the sponsor, it is time for the team to talk to the data architect in charge of managing the data the team needs to start their tests. Again, there are several low-level options that need to be specified,

such as the granularity of the gathered data and the way they will be transferred, whether it is periodically or just in a single transference. If the team is going to need access to the company's infrastructure, user permissions should be created on this phase, too. There are two kinds of variables, depending on their possible values:

- **Continuous variable:** these variables take the form of numbers, being able to take a value on the whole spectrum of real numbers or on definite intervals. Common continuous variables could be a person's height, the annual income of a company or the amount of energy consumed by a certain population.
- **Categorical variable:** these variables can only take several specific values. They are used to model qualities that can't be measured with a number. Common categorical variables could be a person's nationality, the business sector a company works in or the source of the energy a city consumes. There's also the option of having an ordinal variable, a subtype of categorical variables where the possible values follow a specific order, giving a sense of hierarchy in that variable. Typical categorical variables could be the number of kids a family has, a person's position in a company or her educational experience. There is a special type of ordinal variable called interval variable, where the conceptual difference between different values is always the same. This kind is used generally to divide continuous variables in different bins, such as dividing incomes in blocks of 5000\$ or weights in bins of 10 kg.

Data cleaning: once the data have been gathered and it's on the team's hands, it is time for them to start cleaning and transforming it so it can be successfully transferred into the team's preferred analysis tools. Generally, this task will include parsing the data into a csv file or creating some kind of SQL database for them. After it's been loaded into an analysis tool, such as Python or R, it needs to be examined for data quality, such as missing fields or incorrect values. The strategy applied to these kind of values is highly dependant on the problem and the kind of

variable whose value is missing, but usually it implies interpolating or substituting with the variable's expectation for continuous variables and creating a new missing category, deleting those records or applying any kind of grouping algorithm (manually applying prior knowledge or some kind of clustering technique, such as k-means) to assign to each missing record its most probable value.

Data validation and exploration: this phase is composed of two distinct steps. On the first one, the team analyzes the data to assess that it is: large enough, varied enough and appropriate to solve the problem at hand. It must be noted that all these conditions are subject to the problem that must be tackled and that their verification is mainly a manual, subjective process, even though it can be supported by data statistics. If it didn't fulfill this requirements, the team would have to go to the data architect again, expose the data deficiencies and expose how they could be improved in a next delivery. After confirming that, a priori, the supplied data will be useful, the second step of data exploration begins. Now, it is time to start analyzing the characteristics of each variable in the data and discovering relationships between them. There are two possible ways of doing so: analytically, made by calculating and obtaining statistical values of variables and visually, by creating different plots and charts that highlight different characteristics of a variable or the relationships between two or more variables.

This part serves as a guiding step for the team in order to tackle the next point, as it is on this step when the team discovers correlations and relationships between variables within themselves and with the target. Also, this phase should generate the first deliverable of the project: an exploration report where each variable's features and useful or interesting relationships between them should be exposed, whether it is through numerical or visual analysis (preferably through both of them).

Modeling: on this stage, statistics and machine learning finally make their appearance. Here, the team applies their techniques to, hopefully, be able to extract useful, generalizable information that will solve the proposed problem. The actual modeling process requires a previous step that, even though it's not precisely modeling, it is deeply associated with it: the transformation of data to comply with

each specific method's required input. This is a necessary step, as, depending on the software used, each method will require a different transformation. After the data are transformed as required, the actual machine learning starts. There are several typical tasks commonly seen in data science modeling:

- **Classification:** to decide if the objective data belongs to one category or another. Applied to categorical, ordered or interval variables.
- **Scoring:** an analogous task to classification, applied to continuous variables. It consists in estimating the value of the target variable.
- **Ranking:** to order items according to certain conditions.
- **Clustering:** to group instances of data based on their similarity according to the available data.

As it is quite common that the team will try several approaches and different methods to try and extract useful information, they will be going back and forth between the different steps that compose the modeling phase, namely: deciding on which model to use, based on the team's expertise and the problem to solve; adapting the data to the decided method required input; training the method (generally, in the context of a data science project, this step is hidden by software libraries, which do this task by themselves, transforming the training method into just a single function call) and evaluating the obtained results. This is usually done through the computation of specific metrics, commonly used in the context of data science. Depending on the predicted variable's nature and the task at hand, there are different kinds of evaluation methods:

Scoring task: the most often used evaluation metric is the mean squared error, defined as $MSE = \frac{1}{n} \sum_{i=1}^N (y - \bar{y})^2$ where y is the real target variable value and $\bar{y}$ the estimated value by the method. The reason to use a squared error instead of just average difference is that the squared error is the Maximum Likelihood Estimator [10].

Classification task: when the predicted variable is categorical, all the metrics are extracted from what's called a confusion matrix, that contains the four possible different outcomes for a classification problem:

Predicted Value \ Real Value	True	False
True	True Positive (TP)	False Positive (FP)
False	False Negative (FN)	True Negative (TN)

From these four quantities, a whole array of different metrics can be calculated (please note that the confusion matrix form is made for binary classification problems. In a multiclass classification problem there is a confusion matrix for every possible class the output variable can take). The first one that would come to the reader's mind is the accuracy, that measures the percentage of time the method predicted the correct outcome. But there are several others, because the team may be interested in different behaviours. Specifically, there are two pairs of metrics that are very common:

- Precision and recall: defined as $\frac{TP}{TP+FP}$ and $\frac{TP}{TP+FN}$ they measure, respectively, the percentage of time the estimator is right when it forecasts a TRUE value (precision) and the percentage of TRUE values from the real output that the estimator correctly predicts as TRUE, or true positive rate (recall).
- Specificity and sensitivity: specially used in medical contexts, specificity is defined as $\frac{TN}{TN+FP}$ and it represents the true negative rate of the estimator. Sensitivity is just another name for the recall metric we defined on the previous point.

There are several other metrics derived from these, such as the F-score, which joins together the precision and recall values for a single way of scoring an estimator (calculated as $F1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$, or the area under the receiver-operator curve (ROC curve)[10], commonly known as AUC, which is fairly more complex than these. It is calculated by plotting a curve of the False Positive Rate (or

$1 - \text{specificity}$) against the sensitivity of the model as the threshold for issuing a positive prediction is changed over all its possible values. Then, the area under the resulting curve is calculated. Every metric I've talked about is ranged between 0 and 1, where, specially for the F1 score and the AUC, 1 would mean a perfect model, whereas 0 (or even values lower than 0.5 for the AUC) are bad-performing models.

Additionally, in order to ensure that the obtained knowledge is generalizable, a process known as cross validation is followed. It consists in splitting the available data in k different folds, or chunks, and then training the desired model k times, each time using as training data a distinct set of $k - 1$ of the available folds, evaluating the model against the other fold. The obtained metric is then averaged and serves as a good estimation of how the model would work with new data, which is the ulterior objective of training a model. This process, called k -folds cross validation, is the most common one, though there are more types of validation schemes.

Ordering task: an ordering task with more than two levels is just a multivariate classification problem. As such, the techniques we described in the previous point apply here too.

Clustering task: there are a lot of different metrics to evaluate clustering methods, but they can all be classified in two different types: internal evaluation, when the metric applied uses the clustered data itself to evaluate the results, generally by applying a similarity measure and scoring highly methods that produce clusters with high similarity in their data and high dissimilarity between intercluster data, and external evaluation, when the method is evaluated against information not used in the clustering, such as a gold standard created by domain experts. Machine Learning will be defined and explained extensively in the next chapter.

Results documentation and presentation: after the team obtains a suitable model that solves the proposed problem, it is time to prepare the results to be presented.

1.1.3 FOLLOWED METHODOLOGY

Now that I've shown which research frameworks have influenced my work, I'll show the specific steps I've followed to carry out the research this thesis contains, as well as map it to chapters or parts of it in order to ease their understanding and illustrate how I followed them. Fig. 1.1.5 contains these steps. It is clearly an adaptation of the Design-Science Paradigm to the situation of a thesis where its research has been carried out close to the industry. Now, for the Problem Identification and Solution Requirements Definition steps, I performed a deep, thorough review of the scientific literature on the Failure Management issue. When further open problems arose (the ones treated in Contributions 2,3 and 4), I also reviewed the literature for their relevant topics. This review can be seen on Chapter 2. Regarding the development of a solution and its evaluation, the chapters can be divided in two different groups. For Contributions 1 and 4, I leveraged the existing knowledge base, utilizing well known algorithms (for Contribution 1) and standard Big Data Software pieces (Contribution 4) in order to design my solution. Contributions 2 and 3 were different, though. For Contribution 2 there were no previous efforts to propose a general method to address the issue it deals with, so I had to develop the problem formulation and solution from scratch. Contribution 3, though, had some previous work that solved the issue, but it requires human expertise. As one of my objectives is to reduce the need for human expertise or interaction, I propose an alternative solution that deals away with it. These developments are found on Chapters 5, 6, 7 and 8. As for the dissemination of my results, Chapter 5 is supported by references [11] and [12], Chapter 6 is under review [13], and Chapters 7 and 8 were published on the IEEE Communications Magazine [14][15].

Lastly, and to further illustrate how all of my research has been sparked by real, industrial needs, I'd like to talk about the scenarios and projects in which I took part. The genesis of my thesis started on 2013, when I started working at the Center for Open Middleware, a joint research center by the Universidad Politécnica de Madrid and Banco Santander. There, I joined the team for the Online Failure Prediction project, where we would get data from their network managers and

try and develop solutions to predict failures on their networks. This is where my first dataset, the Networking Dataset, came from (both will be described in detail later). Working on these solutions was when I detected the open problems that my research deals with. The work I’ve done from 2015 to 2018 has dealt, specifically, with creating the solutions I expose on this thesis. Furthermore, the experience and Data Science knowledge I’ve acquired by working on them has allowed me to work in several different research areas, where the methods I applied to networks were perfectly applicable. First, in 2017 my team and I collaborated with Fujitsu Spain, helping them developing a failure prediction model for Wind Turbines. Additionally, I’ve collaborated in applying Data Science techniques to Wi-Fi location data. This has spun a new research line in our group that, at the moment, is still in its infancy but is very promising [16]. Along this, I also conducted a Doctoral Internship at the National Institute of Informatics in Tokyo for three months, from March 2016 to the end of May 2016. There, I worked under Professor Yusheng Ji on the “Big Data for Disasters” project, where I applied my Data Science knowledge and skills to a dataset of the population in Tokyo extracted from cellphone antennae readings.

Figure 1.1.5: Actual methodology steps I followed on my research.

1.1.4 HARDWARE AND SOFTWARE USED

Table 1.1.1 shows which software and hardware systems were utilized for each contribution. System A refers to a cluster of eight HP ProLiant SL250s Gen8 with the following specifications: two Intel Xeon e52630v2 2.6GHz (6 cores each) and 32GB of RAM. System B refers to a Macbook Pro 2014, with an Intel i7 2.8 GHz processor (4 cores) and 16 GB of RAM.

Table 1.1.1: Hardware and software utilized for each contribution.

Contribution	Hardware system	Main Software
1	A	R programming language
2	A	R programming language and Spark
3	B	R programming language
4	A	Spark

Next chapter will describe each software in detail, as well as give an overview of the current state of the scientific literature and research on the environments this thesis deals with.

2

State of the Art

As the Introduction chapter served as an overview of my research as a whole, this one does a similar thing to the research landscape on the scientific disciplines I worked on. As such, their structures are very similar. I first presented the problem, then justified its importance and gave a general idea about the open problems on it and my solutions. This chapter is a parallel of it, but focusing on studying the scientific literature that leads me to those conclusions and analyzing it. Thus, I will first explain the works done in the area of Failure Prediction, including the standard way of analyzing temporal event data, divided by the different application areas it has been applied to. After that, I'll summarize its current state, emphasizing the presence of any still unsolved problem. That will lead to a section in which I'll analyze the state of each problem, giving an account of how these problems are currently treated. Lastly, I'll give a scientific foundation of each solution I proposed, describing and justifying each tool I used for each task.

2.1 FAILURE PREDICTION

The best summary of the general state of (Online) Failure Prediction methods is given by Salfner, Lenk and Malek on [17]. It gives us several main ideas that can be used to accurately frame my research. First, it formalizes the concept of Online Failure Prediction as "the identification during runtime if a failure will occur in the near future based on the monitored system current state". Thus, it is different from system reliability prediction in the sense that the former deals with a short time ahead (in the range of minutes, normally) whereas the latter studies long-term metrics, as architectural properties. It also gives a standard vocabulary for the task of Online Failure Prediction, that is shown graphically on Fig. 2.1.1.

Figure 2.1.1: Basic Online Failure Prediction concepts.

It shows four values that need to be defined:

1. Δt_d : this is the time period that is considered in order to perform the prediction. Along this work I'll call it Observation Window.
2. Δt_p : the time period during which the issued prediction is valid. I'll call this the Prediction Window.
3. Δt_l : the Lead Time for a given prediction, separating the Observation and Prediction windows.
4. Δt_w : the minimal warning time. This is a value related to the system and the current ways to address failures. It is determined by the response capabilities of the network managers. Δt_l must be larger than this value.

This theoretical formalization gives us the idea that only recent events should be considered. It doesn't give us, though, which kind of preprocessing should be done to the temporal data chosen within the Observation Window. This falls into the realm of Feature Engineering, a part of the Data Science process that deals with shaping the input data to the prediction models in order to include certain information or maximize the model's performance. For example, a common task is to include interaction terms: the effect of certain variables changing at the same time. In the case of Online Failure Prediction, a transformation must be defined that takes the sequential, input data, and turns it into "clean data", understood as data tables where rows represent a single observation and columns a single variable, which can be fed to a Machine Learning algorithm. The specifics of this function depend, as can be expected, on the nature of the sequential data: numeric variables will be summarized in a much different way than categorical values. Chapter 4 details the specific transformation I used for my datasets.

Now that we have an intuition about what Online Failure Prediction is and some of its concepts, we can move on to study how it has been dealt with on recent scientific literature. While [17] contains a thorough review of different works and methods, the area of Online Failure Prediction suffers from a problem right from the start: there is no public benchmark that is utilized as a reference between all researchers to compare the effectiveness of different methods. Of course, this can be somewhat understood: the nature of the analyzed dataset can be as varied as the different possible systems and networks. It is hard, though, then, to find a good gold-standard to utilize. Nevertheless, this has been done in cousin areas to Distributed Systems Failure Prediction, such as Network Anomaly Detection, on [18] and [19], in which the authors compare several algorithms in terms of performance and computational complexity, against several common datasets. So it should be something that my research area should strive to, as, without common grounds, the research landscape turns into islands, separated from each other. On the rest of this section, I'll go over the most relevant research works done on the area of Failure Prediction on Distributed Systems.

2.1.1.1 DISTRIBUTED SYSTEMS

Research on this area can be divided, broadly, in two different categories: computer networks and computing clusters, such as supercomputers or cloud centers. On the network side of things, there have been both theoretical and practical works. For example, [20] defines a management architecture for large-scale networks based on pattern recognition through sequence mining, and [21] proposes uniFaFF, another framework for autonomic fault management and detection. Nevertheless, none seems to have caused a great impact on the management landscape. On the practical side of things, current works can be divided in two families, based on whether the performed prediction is supervised (this is, the data used has been previously labeled with failures) or unsupervised (the data has not been labeled). Some of the research papers based on supervised learning are: [22], where the authors propose a three-phased method based on clustering event sequences, finding intra-cluster dependencies and creating rules between events; [23], in which the authors use their own real network logs to predict when a failure will happen using Bayesian Learning; [24], where the authors propose as a novel contribution the idea that a failure will present burstiness and then define three functions: the creation of log templates, the extraction of feature vectors from these and the training of a Machine Learning model, validated using realistic log data from a large network. Additionally, [25] utilizes fourteen months of their own network logs to test their method based on creating observation windows, optimizing model parameters and training a prediction model, validating against a Weibull distribution; [26] compares the effectiveness of Hidden Semi Markov Models against three other prediction methods, utilizing field data of a commercial telecommunication system and [27] presents the Opprentice system, in which network managers first label anomalies and then a Random Forests model is grown to predict them. Conversely, I found more research papers that utilized unsupervised learning. This is, mainly, due to the fact that network-related labeled data is scarce and hard to obtain. The application of this kind of models on Failure Prediction tasks is called Anomaly Detection. Some of the main works are

on [28], which detects failure patterns by removing redundant data and then performing clustering on them, using large production data as validation; [29], where the authors correlate failure appearances with system metrics; [30], where Principal Component Analysis is applied to detect anomalies; [31], which also presents a tiered system for syslog mining: first, they represent the system's behavior as a mixture of Hidden Markov Models, then they train a prediction model that gives an anomaly score and, lastly, [32], where the authors apply eigen equation compression to build network and node-level features for the models, calculating then an anomaly score based on the probability distribution of the built features. The main conclusion I extract from this category of research is that there is no public, labeled dataset that serves as an anchor for the community, allowing scientists to compare their works. Everyone uses their own logs and datasets. That isolates them and is something that we should strive to solve.

Moving on to cluster systems, we also find a plethora of techniques applied to them. [33], for instance, applies the Auto Regressive Moving Average method for time series prediction and Fault Tree Analysis to predict failures on a simulated cluster based on the Simi platform. [34] classifies event messages by their similarity, finding patterns on them, which is similar to what [35] proposes, but they go one step further: after mining correlations, the authors apply apriori-like algorithms to logs parsed as n-ary sequences where each event is a nine-tuple. In a similar way to what happened to networks, there is also a lot of work done on the Anomaly Detection part. Several interesting conclusions are found, for example, on [36], which presents an anomaly detecting system, auto-aid, through outlier detection, and is validated on a 362 high computing cluster institution grid and on [37], which utilizes their own cloud dataset to test a mixture of unsupervised (to label) and supervised (to predict) methods. Differently to networks, though, there are several public quality datasets found on the High Performance Computing area which have been extensively tested. For instance, the Ranger Supercomputer on Texas has been utilized on [38], in which the resource usage monitor TACC_Stats is used to detect resource usage anomalies, and on [39], which presents a decentralized clustering algorithm to detect anomalies. The Blue Gene implementation

of supercomputers also deserves mention, its L and P versions being the subject of several interesting research works. Namely: [40] shows a Failure Prediction method based on a Genetic Algorithm, [41] applies three simple methods to predict failures based on temporal and spatial correlation, [42] studies the influence between performing period-based prediction (having a Prediction Window larger than o) and event-driven prediction (punctual prediction, predicting exactly the moment of appearance of an event) and two more papers where Blue Gene is utilized alongside more datasets: [43], where the authors test Support Vector Machines, sequential rules and Hidden Semi Markov Models against different Observation Windows and [44], in which the authors propose three preprocessing steps to improve prediction's accuracy: event categorization, temporal and spatial filtering and causality-related filtering to create association rules between events. Another interesting dataset is the one coming from the Los Alamos National Laboratory, described on [45], which is publicly available and feature rich. In fact, it is one of the datasets I chose to utilize for my experiments. It will be described in detail in Chapter 4.

I would also like to remark that there have also been research efforts on other kinds of distributed systems, such as [46], that applies probabilistic graphical models to predict time series values on a web services dataset and [47], which performs a combination of Random Indexing and Support Vector Machines on a dataset from a large European manufacturing company.

2.2 OPEN PROBLEMS

Overall, I draw the following conclusions from my review: broadly, and specially on certain areas (mainly, Computer Networks), research is done on a vacuum and, specially for supervised Failure Prediction, there is little to none work done with public datasets and with more aims than just solving a specific, ad-hoc problem. There is no comparison of methods, just the application of a good-enough idea. And, as there are no standard public labeled datasets, proposals can't be compared. I also found three more problems. First, there is a serious lack of study on the effect

of choosing an optimal Observation Window. As most problems are solved utilizing private data, researchers tend to know them profoundly, but nevertheless, just setting an arbitrary value is hardly ever good enough. As a second problem I detected a lack of integration between prediction methods and causal analysis: knowing why an event happened is as important as knowing if it will occur, and most proposed systems don't consider the former. Lastly, the architecture proposals that I found are theoretical, not implemented on reality. There is not a realistic proposal of how to bring the Online Failure Prediction concepts to an industrial system utilizing standard software pieces. On this section I'll go over each of the three problems, analyzing what has been done to address each of them.

2.2.1 OBSERVATION WINDOW SETTING

The problem of determining an appropriate Observation Window is pervasive on literature: every author that deals with observing the past to predict the future must decide which time period to consider. In most of the works I reviewed on the previous section this problem was present and mostly none of them acknowledged it, just selecting a single value to use. To further study the current state of this problem, I performed another search and overview of the present Failure Prediction literature, this time focusing on how the authors treated the window issue. In case there were potential interesting solutions in other disciplines, I did not restrict my search to distributed systems, though they were my main focus. Dividing, then, the papers I studied based on the application area, there are three different subsections:

SUPERCOMPUTERS AND HIGH-SCALE CLUSTERS

Overall, this area was the most frequent. I observed varied coverage of the window problem. The only papers I found that directly addressed it regarding Failure Prediction were [43], where different observation windows (1, 5 and 12 hours) are tested in order to predict failures on cluster systems, [42], in which the authors compare period-based prediction vs event-based prediction and study the influ-

ence of several observation windows and lead times on both, and [48], where the effect of the observation window size in the uncertainty of failure predictions is studied. In contrast, the framework proposed on [38] employs a fixed window based on the user’s knowledge. A similar issue is present on [46], where the usage of a Dynamic Bayesian Network is proposed for reliability analysis on service oriented systems but only use a fixed observation window of 20 seconds. The observation window problem arises in several other works in this area, though not applied to Failure Prediction: [22] mentions how assuming a fixed window is not convenient for performing Root Cause Analysis and [28] studies the problem of learning patterns in failure logs of distributed systems, in which the time between failures is a key parameter to tune. I consider these issues to be close to my own problem. Lastly, there is another research [49] that proposes a solution for a problem analogous to mine, in the field of log filtering and aggregation.

COMPUTER NETWORKS

In [25], Zhong, Guo and Wang compare four different methods (RIPPER, BayesNet, Random Forest and a Weibull distribution) with four different window schemes (90 minutes, 60 minutes, 80 minutes in two windows and 30 minutes in three windows). There is a similar paper in which network logs are analyzed in six different chunk sizes[24]. These are papers in which the time problem is studied. Conversely, in [30] a 1 hour fixed window is assumed for detecting network anomalies using Principal Components Analysis (PCA) and in [50] RCA is performed by clustering samples of events without studying the effect of time on it.

HARD DRIVES AND OTHER AREAS

A similar discipline in which this problem is also present is predicting failures in hard drives. Though I found only three research articles about it, all three assess the observation window’s effect on their methods’ performance: [51] tests Neural Networks and Support Vector Machines in four different observation windows, [52] applies Classification and Regression Trees against an array of six observation

windows and [53] proposes the usage of Gaussian Mixture Models, testing it with several samples. Neither in this area nor in previous ones were any general conclusions extracted, which supports the idea that each dataset and system requires specific knowledge about it.

Apart from large-scale systems, networks and hard drives, the observation window problem has also been found on areas as diverse as software systems [54][55], theoretical analysis of temporal data [56][57], the internet of things [58] and even sensor networks on railings [59].

2.2.2 ROOT CAUSE ANALYSIS

The second open problem I tackle deals with Root Cause Analysis (RCA), which is key to prevent failures and errors in networks and systems and to ensure downtime is kept to a minimum when they happen. It deals with the identification of the causes of errors, taking an integral part in network management systems along with monitoring, prediction and reparation. As I talked about while analyzing the current state of the art, most RCA systems and prediction methods are isolated, which is a waste of potential, as there is information to be extracted and shared from both.

As stated in [60], the main techniques for RCA include searching the issued problem into a knowledge base -usually organized into sets of rules- to obtain its cause. So, it is possible to find: case-based reasoning, where the base is a library of errors described by their cause and past solutions; model based systems that hold models of behavior for each device in the network; or collections of system events labelled either symptoms or problems in a correlation matrix (codebook). Whenever a problem happens, information about the system state is checked against the knowledge base to take a decision to solve the failure.

Creating the knowledge base is usually difficult, as it mainly relies on network operator's knowledge and is based on rules defined by them, extracted from their experience working on the system. Thus, it is expensive to create and maintain, and completely specific to the system it was created for. Even so, this kind of ap-

proach was good enough for small networks and static systems. But, as we already know, current networks are large and extremely complex, which renders any system based primarily on human knowledge incapable of completely covering all the changing network properties and features [60][61][27].

Bayesian Networks (BN) are the base of most used RCA methods so far [61]. They model system features such as events, conditions or metrics, as nodes in the BN, and their dependencies are expressed as conditional probability. Then probabilistic inference is used within the dependency model to obtain the most probable root causes of errors. This method scales well and can adapt to dynamic environments, but it is not without hindrances: it suffers of performance issues (though there are efforts that partially solve this [60]), do not perform well on high dimensional data and are extremely dependent on prior knowledge and the distributions chosen to model the data.

Just to alleviate these problems, research has been varied in scope and techniques used. For instance, in [62] the authors tackle the problem of data variety, which is one of the issues the size of current networks brings along. To address this, the authors propose to aggregate information into message templates and effectively reduce the number of event categories to deal with. Then, frequent patterns are found through data mining, which creates a database for a case-based reasoning-like system.

Some of the other open issues on RCA have also been recently explored through improvements or combinations of methods. A perfect example of these efforts is the work found on [60], which implements an RCA system that solves the efficiency problem by joining Bayesian Networks with case based reasoning: the network is only used with new cases. The same authors had previously presented in [61] a different solution for the same problem, this time by clustering the network on different groups, which allows them to shrink the nodes to analyze, effectively reducing the inference time.

All in all, the problem of creating the knowledge base in an understandable form remains; so, Machine Learning algorithms are also having their share of the spotlight. For example, Random Forests, the algorithm I will use in my solution for

this problem, has already been applied successfully in [27], where the authors feed it with two different sources of data: first, network operators label network data where they find anomalies, second, they run a series of anomalies detectors previously proposed on literature over the same data. From these two sources, they create a labeled dataset which is then fed into the Random Forests algorithm, which then constitutes an Anomaly Detector on its own. While this approach is thorough and very interesting, it assumes the collaboration and knowledge of network operators and managers is available, and that goes in contradiction to the features that a future management system should have.

Lastly, one emergent technique that could become very important in the next years for understanding complex systems is summarization, understood as expressing a system in reduced, but valuable, information. One of the most used techniques for this task is clustering, a data mining technique based on grouping the data based on their features. This way, all the data can be summarized just by giving some cluster statistics. This technique was successfully applied to Root Cause Analysis on [50] in the context of an IT environment. Another interesting usage of summarization is shown on [63], applied on social networks graphs.

Despite the efforts in autonomic management [64], the response to critical situations is set by human judgement -at least in the industrial setting- so understanding the root causes of errors and expressing them in a clear way is of paramount importance for the practical application of RCA methods.

2.2.3 BIG DATA IMPLEMENTATIONS

The third problem I pointed out was a lack of practical Online Failure Prediction implementations based on Big Data technologies. I opted directly to Big Data implementations due to the sheer velocity, computation capacity and storage size needs that future Management Systems demand. When I moved to study in detail the current research landscape in Big Data solutions, I found four areas. To understand them, we need to consider that Big Data, understood as the processing of large quantities of data, is a very recent discipline and, mostly, an industrial

one. As such, it has taken the academic side of research almost by surprise, except for certain luminaries, and most works have been, indeed, reactive. As such, the biggest, by far, kind of scientific work I found when searching for Big Data research was surveys and overviews indicating the current situation of the discipline, its challenges and future lines of work. For example, [65] focuses on the aspect of reliability management, which is a close discipline to my own. It gives several interesting observations, such as the need for data integration from different sources, the importance of working with realistic data instead of synthetic ones and some of the tasks that Big Data processing allows, such as predicting the lifetime of a system or a cluster of systems or the maintenance costs. [66] focuses on the need of establishing a systematic approach to handle and analyze data as a requisite for, as it calls it, Industry 4.0 Management systems, which should add self-awareness and self-predictiveness to current features, such as condition monitoring and fault diagnosis. It also gives several theoretical examples of possible Big Data applications. Another feature that future systems should aspire to have, and it is one that self-* properties allow, is transparency [67], which is close to the concept of autonomy and eliminating human interaction I previously outlined. Overall, most of the surveys deal with the same ideas and issues, and there has been a huge number of them in recent years [68][69][70][71][72]. Some more unique articles are [73], which focuses on working with online data and gives several requisites and features that a streaming data processing system should have, and [74], where the authors, apart from giving a solid review of the different approaches, mechanisms and frameworks for Big Data, define four modules as part of a systematic framework to perform this kind of computation. I have two remarks to make on this part. The first one is that almost every paper that I've cited suffers from the same problem: obsolescence. Being such a young and rapid-changing discipline, surveys from three years ago can portray inaccurate information today. And, second, I did not detail each paper specifically not because they are not important or distinct, but because a general foundation for Big Data will be given on the next section, and the focus of this subsection is not to explain in detail every survey and challenge, but to understand how the current research corpus is related to Online

Failure Prediction implementations.

I mentioned four different kinds of papers on this area. Surveys and overviews were the first one. The rest are closer to practical implementations, but much more infrequent. The second one is a series of papers talking about Spark [75], the current de-facto framework for Big Data Analytics, which has superseded Hadoop [76], and its capabilities for some applications, such as smart grids [77]. This framework is the basis of my own proposal and will be detailed in the next section. I also found several papers comparing different Machine Learning implementations [78][79], in which Spark's own MLlib emerges as the recommended toolkit. Lastly, regarding actual Big Data implementations of something similar to an Online Failure Predictor, I only found two relevant bodies of research. [80] proposes an alternative stream processing framework and programming model in which an Anomaly Detection system is implemented. It's important to remark that one of the reasons they do so instead of using, for example, Spark Streaming, is because they argue that the operators available to user code is very limited. This issue is no longer present as of 2018, as such, Spark Streaming is still an optimal candidate and the de facto industry standard. And, lastly, the only work I found actually using Spark was [81], which presents a similar implementation to mine, using Spark and Kafka. There are two main differences between my work and the one in this paper, though: they perform Anomaly Detection based on clustering while I perform Failure Prediction and they validate their proposal using synthetic data obtained from VMWare, while I use real data obtained from an industrial scenario.

In conclusion, the Failure Prediction research area still needs a practical implementation of the concepts addressed in research papers to show and validate how they apply to the real world and can be deployed in real industrial environments.

2.3 PROPOSED SOLUTIONS AND THEIR FOUNDATIONS

On the first section of this chapter I analyzed the general state of research on Distributed Systems Failure Management and extracted several open problems from it. Then, on the next section, I delved deeper on each of them, showing how and

what previous research had been done for them, showing how they are still relevant and not addressed. In this last section I will give the theoretical foundations and justifications for the tools I will use on my proposals to tackle each problem, which will be detailed on next chapters. I will give an overview of several Machine Learning methods to give a foundation to most of my experiments, then justify the usage of a Genetic Algorithm for the Observation Window problem and, lastly, I'll give an overview of Big Data and its current, main software components.

2.3.1 MACHINE LEARNING

Machine Learning is a discipline derived from the intersection of Artificial Intelligence, Computer Science and Statistics which has several definitions. Alpaydin[10] defines it as "programming computers to optimize a performance criterion using example data or past experience". Tom Mitchell [82] argues that, to define a scientific field, one should see which is its central question. Following this logic, he states that machine learning seeks to answer "How can we build computer systems that automatically improve with experience, and what are the fundamental laws that govern all learning processes?" and clarifies: "we say that a machine learns with respect to a particular task T , performance metric P , and type of experience E , if the system reliably improves its performance P at task T , following experience E ". Taking a middle ground between Alpaydin's simple definition and Mitchell's practical one, I offer the following definition: Machine Learning is a scientific discipline that studies how to program software to correctly predict an expected value based on available data and improve its performance by learning from more data.

We can see, then, that it is through Machine Learning that a Data Science team is able to generate the solution for the problems it faces. The core objective of Machine Learning is to be able to infer, from a sample of data, patterns or regularities that are applicable to the whole data population. One key idea of Machine Learning is the fact that it is applied usually in environments where it is impossible to create definite rules or define the underlying function completely, where the amount of data allows to create a good enough approximation of the desired

output. Even though it may seem that I'm speaking about Statistics, and it would be understandable to think so, let's not forget Machine Learning is an intersection of Statistics with one more discipline. Computer Science fills two different roles that differentiate Machine Learning from Statistics [10]: while training, it's common to generate a huge amount of data which needs to be stored and processed. Data Management is, then, a key component of Machine Learning. Furthermore, the algorithmic implementations of the techniques used to train and compute the model's output need to be translated to code with the highest efficiency available. These tasks are both in the field of Computer Science. It is also considered a sub-field of Artificial Intelligence because, obviously, it is the part of it that adds the equivalent of human learning to computer algorithms (this expression should be taken with a grain of salt, due to the enormous differences between one and other, furthermore taking into account how little we know about human learning). Machine Learning has been applied successfully in a whole array of different contexts. In fact, overall through the literature review I performed, most of the papers I talked about were performing some kind of Machine Learning training at some part.

CLASSIFICATION

There are several criteria by which machine learning algorithms can be classified. One of it, presented by Ayodele [83] is the "desired outcome of the algorithm". Based on it, and the ones found in [9] and [10], I present a classification of the algorithms based on two parallel axes. The first one is the nature of the data used and the process by which the algorithm is trained:

- Supervised Learning: where a function is generated that maps an input to a desired output. These inputs and outputs are given to the algorithm as labeled data. These labels allow for the establishment of error and evaluation functions that allow the algorithm to tune itself and improve its performance.

- Unsupervised Learning: also called "letting the data speak for themselves", it models or finds structures in the input without any label or desired output.

The second criteria is the actual task the algorithm carries out, or, analogously, its output:

- Classification: a classifier maps a certain vector of inputs to a certain output class or category, a discrete variable. The most used ones are binary classifiers, where the possible outputs are true or false. Multiple output classifiers can be seen as an ensemble of binary classifiers.
- Forecasting: closely related to the statistical method of regression (as the main method for this task is that same process), the output of a forecasting method is a continuous variable, predicted as a function of the inputs, usually weighted by some constants.
- Clustering: these methods try to group similar instances of data in different clusters, based on a user-defined metric of similarity.
- Dimensionality Reduction: the purpose of these methods is to analyze the available instances of data and represent them in less dimensions than the original ones with minimum loss of information in the process.
- Rules Extraction: this method discovers relations between variables which are presented as rules that can be used to forecast the appearance of a variable based on the presence or absence of other ones.

I will now detail some the most popular and used Machine Learning algorithms and concepts.

REGRESSION

Regression is the first and most usual statistical method used in Machine Learning to establish correlations between a dependent variable and an independent one,

model the relationships between them and predict outcomes based on these models. They are supervised learning methods which can be used for classification and forecasting. The general function for regression is:

$$r = f(x) + \varepsilon \quad (2.1)$$

where $f(x)$ is an unknown function approximated with an estimator $g(x|\theta)$, being θ the parameters of the model. The model is optimized by means of least squares estimation, this is, minimizing the error function

$$E(\theta|X) = \frac{1}{2} \sum_{t=1}^N [r^t - g(x^t|\theta)]^2 \quad (2.2)$$

The independent variables can be discrete or continuous. The way of integrating discrete variables in the model is to create as dummy variables as possible values of each discrete variable. Then, each value gets assigned a parameter, which weights the presence of each concrete value. Depending on the kind of relation assumed in $g(x|\theta)$, there are different kinds of regression. Given a set of inputs $X = x_1, x_2, \dots, x_N$ and parameters $W = w_0, w_1, w_2, \dots, w_N$, the most usual ones are:

Linear Regression: in this kind of regression, the assumed relation between the independent variables and the dependent one is linear, and the parameters are constants that multiply the inputs. The univariate estimator's equation is:

$$g(X^t|W) = w_0 + w_1 x^t \quad (2.3)$$

This kind of regression is, even if it is the simplest one, deceptively powerful, being able to model non-linear relationships which a priori wouldn't be modeled correctly by this method.

Polynomial Regression: a generalization of linear regression, used to add non-linear relations between the independent variables and the dependent one. The

equation for the estimator in a univariate case turns now into:

$$g(X^t|W) = w_o + w_1x^t + w_2(x^t)^2 + \dots + w_j(x^t)^j \quad (2.4)$$

Both linear and polynomial regression are solved in an equal way. For a reason of simplicity, I'll only show the method for univariate linear regression. Starting from the MSE equation, substituting $g(x|\theta) = xw_1 + w_o$, we obtain:

$$E(\theta|X) = \frac{1}{2} \sum_{t=1}^N [r^t - xw_1 - w_o]^2 \quad (2.5)$$

Taking the derivative of the equation with respect to the parameters we obtain:

$$\sum tr^t = Nw_o + w_1 \sum tx^t \quad (2.6)$$

$$\sum tr^tx^t = w_o \sum_t x^t + w_1 \sum_t (x^t)^2 \quad (2.7)$$

writing these equations in matrix form we get:

$$\mathbf{A} = \begin{bmatrix} N & \sum_t x^t \\ \sum_t x^t & \sum_t (x^t)^2 \end{bmatrix}, \mathbf{w} = \begin{bmatrix} w_o \\ w_1 \end{bmatrix}, \mathbf{y} = \begin{bmatrix} \sum_t r^t \\ \sum_t r^t x^t \end{bmatrix} \quad (2.8)$$

which can be solved as $\mathbf{w} = \mathbf{A}^{-1}\mathbf{y}$.

Logistic Regression: the most important member of a category of discriminants called generalized linear models, logistic regression is the go-by approach to classification problems. Again, for the sake of simplicity, I'll show the equations for a binary classification case. In logistic regression or discrimination, the basic relation from which we start is

$$\log \frac{p(\mathbf{x}|C_1)}{p(\mathbf{x}|C_2)} = \mathbf{w}^T \mathbf{x} + w_o^o \quad (2.9)$$

Applying Bayes' rule, we get

$$\text{logit}(P(C_1|\mathbf{x})) = \log \frac{P(C_1|\mathbf{x})}{1 - P(C_1|\mathbf{x})} = \log \frac{p(\mathbf{x}|C_1)}{p(\mathbf{x}|C_2)} + \log \frac{P(C_1)}{P(C_2)} = \mathbf{w}^T \mathbf{x} + w_o \quad (2.10)$$

where $\text{logit}(x) = \log \frac{x}{1-x}$, $w_o = w_o^o + \log \frac{P(C_1)}{P(C_2)}$

and rearranging terms, we finally get as an estimator of $P(C_1|\mathbf{x})$

$$y = \hat{P}(C_1|\mathbf{x}) = \frac{1}{1 + \exp[-(\mathbf{w}^T \mathbf{x} + w_o)]} \quad (2.11)$$

which is the sigmoid function, defined as $\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$. The way this binary classifier works is by defining a threshold value. If the sigmoid output is over that threshold, then we assume the input to class C_1 .

Now let's study how to learn the parameters $\mathbf{w}$ and w_o . Given a sample of two classes, $X = \{\mathbf{x}^t, r^t\}$, where $r^t = 1$ if $\mathbf{x} \in C_1$ and $r^t = 0$ if $\mathbf{x} \in C_2$, we assume $r^t|\mathbf{x}^t \text{ Bernoulli}(y^t)$. We use cross entropy as the error function to minimize, defined as

$$E(\mathbf{w}, w_o|X) = - \sum_t r^t \log(y^t) + (1 - r^t) \log(1 - y^t) \quad (2.12)$$

and use gradient descent [10] to minimize it, due to the nonlinearity of the sigmoid function, with the following update rules:

$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j} = \eta \sum_t \left(\frac{r^t}{y^t} - \frac{1 - r^t}{1 - y^t} \right) y^t (1 - y^t) x_j^t = \eta \sum_t (r^t - y^t) x_j^t, j = 1, \dots, d \quad (2.13)$$

$$\Delta w_o = -\eta \frac{\partial E}{\partial w_o} = \eta \sum_t (r^t - y^t) \quad (2.14)$$

stopping when convergence is reached.

NAIVE BAYES CLASSIFIER

The Naive Bayes Classifier is, as its name suggests, the simplest instance of Bayes' Classifiers. Its core idea is to determine the probability of each new instance belonging to each possible class and then just assigning it to the most probable one. Given a sample dataset $\mathbf{X}_i = \{x_1, x_2, \dots, x_n\}_i, i = 1, \dots, N$ and $C_j, j = 1, \dots, K$ possible classes, they make use of the Bayes Theorem:

$$p(C_j|\mathbf{X}_i) = \frac{p(\mathbf{X}_i|C_j)p(C_j)}{p(\mathbf{X}_i)} \quad (2.15)$$

where $p(C_i|\mathbf{X})$ is the probability that given instance $\mathbf{X}_i$ it belongs to class C_j , $p(\mathbf{X}_i|C_j)$ is the probability of getting instance $\mathbf{X}_i$ given class C_j , and $P(\mathbf{X}_i), P(C_j)$ are occurrence probabilities of instance $\mathbf{X}_i$ and class C_j , respectively. As the denominator is common for every class calculation, we can dispose of it. Expanding the numerator we get:

$$\begin{aligned} p(\mathbf{X}_i|C_j)p(C_j) &= p(C_j, \mathbf{X}_i) \\ &= p(C_j, x_1^i, x_2^i, \dots, x_n^i) \\ &= p(C_j)(x_1, \dots, x_n|C_j) \\ &= p(C_j)p(x_1|C_j)p(x_2, \dots, x_n|C_j, x_1) \\ &= p(C_j)p(x_1|C_j)p(x_2|C_j, x_1)p(x_3, \dots, x_n|C_j, x_1, x_2) \\ &= p(c)p(x_1|C_j)p(x_2|C_j, x_1)\dots p(x_n|C_j, x_1, x_2, x_3, \dots, x_{n-1}) \end{aligned}$$

Here is where the naive classifier gets its name: we take the assumption that each feature's value is independent of the other ones, $p(x_i|C_j, x_l) = p(x_n|C_j)$ for every $i \neq l$. This is, we assume that each input feature's value is not correlated with the other ones'. Of course, this assumption is not always true for the input data, but even then, the Naive Bayes Classifier usually does a good job. The class probabili-

ties are then calculated as

$$p(C_j|x_1, x_2, \dots, x_n) = \frac{1}{p(\mathbf{X})} p(C_j) \prod_{i=1}^n p(x_i|C_j) \quad (2.16)$$

This assumption greatly reduces the complexity of the required calculations. Building a classifier out of this probabilities is as simple as selecting the class j that maximizes the posterior probability

$$\arg \max_j p(C_k = C_j) \prod_{i=1}^n p(x_i|C_k = C_j) \quad (2.17)$$

Naive Bayes has been successfully applied in lots of environments and is one of the standard tools for data scientists. One typical application of this kind of classifiers is a spam filter, where a technique called bag of words (which eliminates the ordering in the sentences and just counts whether a word appears in a whole text or not) is applied and spam is categorized based on the appearance of common words that have previously appeared in spam tagged messages.

DECISION TREES

Decision trees are a sort of nonparametric, supervised estimation used for classification and regression, created by dividing the input space sequentially in regions defined by a distance measure. After it's built, a decision tree consists of nodes and leaves. The nodes compute a test function and decide the outcome (branch) that the input will follow to. In the end, the tree decomposes a complex decision function into smaller simple decisions. This approach yields two main advantages:

1. Speed: every time a decision is taken and an output is assigned, every other output and its branches is discarded. This increases the speed of finding the correct output.
2. Interpretability: the nodes and branches structure can be easily transformed into a set of IF-THEN rules, which makes the tree's performance very easy

to grasp and also serves as a knowledge extraction method.

:

I will now explain how to construct an univariate decision tree, this is, where only one input variable is taken into account. There are two possibilities for the nodes, depending on the nature of the input. If it is discrete, taking one of n values, the tree performs a n -way split, outputting up to n branches. If the input is numeric, the node performs a binary split against a threshold value. The objective is to build the smallest tree that explains the data with no error or minimum error. Of course, this is a very complex problem (NP-complete) so we rely on local search heuristics. The process of building it is different depending on the objective of the tree: classification or regression. So, I'll explain both separately.

Classification Trees: For classification trees there is a measure to be used to measure how good a split is, and that is the impurity measure. A node is defined as "pure" if all the instances that end up in that node are from the same class. Analytically, we can measure a node's impurity with the following formula:

$$\hat{P}(C_i|\mathbf{x}, m) \equiv p_m^i = \frac{N_m^i}{N_m} \quad (2.18)$$

where m is the node, $\mathbf{x}$ is the input, C_i is the class i , N_m is the total amount of instances that get to node m and N_m^i is the amount of instances that get to node m and belong to class i . Taking this into account, we define a node being pure as one whose p_m^i are either 0 or 1. Another possible way of measuring it is entropy, defined as

$$\iota_m = -p \log_2 p - (1-p) \log_2 (1-p) \quad (2.19)$$

Actually, any function $\varphi(p, 1-p)$ is a purity measure of a node if it satisfies

- $\varphi(1/2, 1/2) = \max(\varphi(p, 1-p))$.
- $\varphi(0, 1) = \varphi(1, 0) = 0$.
- $\varphi(p, 1-p)$ is increasing in p on $[0, 1/2]$ and decreasing in p on $[1/2, 1]$.

Of course, entropy satisfies this conditions, but there are other functions that do too, such as

- Gini index: $\varphi(p, 1 - p) = 2p(1 - p)$
- Misclassification error: $\varphi(p, 1 - p) = 1 - \max(p, 1 - p)$

After computing a node's purity, if it's impure the items on that node are split again to minimize impurity. Analytically, let's define N_{mj} as the number of instances out of N_m that go to branch j at node m . Thus, we can define

$$\hat{P}(C_i | \mathbf{x}, m, j) \equiv p_{mj}^i = \frac{N_{mj}^i}{N_{mj}} \quad (2.20)$$

and the impurity after the split as

$$I'_m = - \sum_{j=1}^n \frac{N_{mj}}{N_m} \sum_{i=1}^K p_{mj} \log_2 p_{mj}^i \quad (2.21)$$

This process continues after every node is pure and can be transformed into a leaf with its associated class. Of course, this isn't always possible as there may be noise. In this cases, the appropriate strategy is to define a threshold purity instead of forcing it to be 0 or 1. This threshold is what defines the complexity of the tree. A low value forces the tree to grow big in order to classify the set correctly and a high one makes a small tree with a large bias.

Regression Tree: the process of building a regression tree is very similar to the construction of a classification tree. The only difference is that the impurity measure is changed to the mean square error from the estimated value at a certain node. Let's take a node m , where X_m is set of instances that reach it, and define

$$b_m(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in X_m \\ 0 & \text{otherwise} \end{cases} \quad (2.22)$$

Being g_m the estimated value, the mean square error at node m is

$$E_m = \frac{1}{\sum_t b_m(\mathbf{x}^t)} \sum_t (r^t - g_m)^2 b_m(\mathbf{x}^t) \quad (2.23)$$

where the estimated value g_m is defined as the mean of the instances that reach the node

$$g_m = \frac{\sum_t b_m(\mathbf{x}^t) r^t}{\sum_t b_m(\mathbf{x}^t)} \quad (2.24)$$

We can see then that the equation for the mean square error is actually the variance of the estimate. Knowing this, we can define a threshold θ_r so when $E_m < \theta_r$ the node turns into a leaf node and g_m is stored as the estimated value. Again, if $E_m > \theta_r$, data in node m is split in a way that minimizes the sum of errors in the branches. Taking X_{mj} as the instances in node m that take branch j , we define

$$b_{mj}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in X_{mj} \\ 0 & \text{otherwise} \end{cases} \quad (2.25)$$

and g_{mj} as the estimation in branch j

$$g_{mj} = \frac{\sum_t b_{mj}(\mathbf{x}^t) r^t}{\sum_t b_{mj}(\mathbf{x}^t)} \quad (2.26)$$

making the error after the split

$$E'_m = \frac{1}{N_m} \sum_j \sum_t (r^t - g_{mj})^2 b_{mj}(\mathbf{x}^t) \quad (2.27)$$

As we saw with classification trees, there are other possible metrics to use when testing nodes. In regression, another popular metric is the worst possible error

$$E_m = \max_j \max_t |r^t - g_m| b_m(\mathbf{x}^t) \quad (2.28)$$

A possible variation is, instead of taking the mean of the instances in each node, to do a linear regression fit in each node.

SUPPORT VECTOR MACHINE

This model starts from the same foundations as the kernel estimator we saw before in non-parametric density estimation and can be used for classification or regression. To show its basis, we'll start by showing how to find an optimal separating hyperplane between two different classes. Let's start with a sample dataset $X = \{\mathbf{x}^t, r^t\}$ where $r^t = +1$ if $\mathbf{x}^t \in C_1$ and $r^t = -1$ if $\mathbf{x}^t \in C_2$. The objective is to find the weights $\mathbf{w}$ that fulfill

$$\mathbf{w}^T \mathbf{x}^t + w_o \geq +1 \text{ for } r^t = +1 \quad (2.29)$$

$$\mathbf{w}^T \mathbf{x}^t + w_o \leq -1 \text{ for } r^t = -1 \quad (2.30)$$

which can be condensed as $r^t(\mathbf{w}^T \mathbf{x}^t + w_o) \geq +1$. It's important to note that we want the output to be separated by a margin, this is, a certain distance between the instances and the hyperplane. To improve the generalization capability of the model, this margin must be maximized. The hyperplane that does this is called the optimal separating hyperplane. The distance of $\mathbf{x}^t$ to the hyperplane is, in our case,

$$\frac{r^t(\mathbf{w}^T \mathbf{x}^t + w_o)}{\|\mathbf{w}\|} \quad (2.31)$$

which we would like to be at least equal or higher to a certain margin ρ . To obtain a unique solution we fix $\rho \|\mathbf{w}\| = 1$ and we set to minimize $\|\mathbf{w}\|$. The problem turns now into

$$\min \frac{1}{2} \|\mathbf{w}\|^2 \text{ subject to } r^t(\mathbf{w}^T \mathbf{x}^t + w_o) \geq +1, \forall t \quad (2.32)$$

As we've seen before, if a problem is not linearly separable we can apply a nonlinear basis function to the data to map it to a space where they are linearly separable. One thing to notice is how increasing the dimensionality of the data can increase the complexity of the problem, so we must find a problem whose complexity depends not on the input dimensionality but on the number of training instances.

As a starting point we rewrite the minimization equation as an unconstrained

problem using Lagrange multipliers a^t

$$L_p = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{t=1}^N a^t [r^t (\mathbf{w}^T \mathbf{x}^t + w_o) - 1] \quad (2.33)$$

which we will minimize with respect to the weights $\mathbf{w}$ and maximize with respect to $a^t \geq 0$. To solve this problem, we will equivalently solve the dual problem, maximizing L_p with respect to a^t , subject to $a^t \geq 0$ and to:

$$\frac{\partial L_p}{\partial \mathbf{w}} = 0 \rightarrow \mathbf{w} = \sum_t a^t r^t \mathbf{x}^t \quad (2.34)$$

$$\frac{\partial L_p}{\partial w_o} = 0 \rightarrow \sum_t a^t r^t = 0 \quad (2.35)$$

If we insert those conditions in the lagrangian problem stated before, maximize it only with respect to a^t and then solve for a^t we see that most of them are equal to 0. The set of input instances x^t whose corresponding $a^t > 0$ are called support vectors and they fulfill the condition $r^t (\mathbf{w}^T \mathbf{x}^t + w_o) = 1$. This condition means they are on the margin. Every remaining instance has $r^t (\mathbf{w}^T \mathbf{x}^t + w_o) > 1$ and does not influence the discriminant. Now, we are able to compute w_o as $w_o = r^t - \mathbf{w}^T \mathbf{x}^t$. This process defines a discriminant called the Support Vector Machine.

It must be noted that, as I said before, the process defined before is only able to separate linearly separable data. If the data is nonlinearly separable, we must allow some deviation from the margin and look for the hyperplane that divides the data with the minimum error. Another option is to apply what is called a kernel function to the data, which projects them into a higher dimensional space in which the data may be separable. The most well known one is the Gaussian Kernel [10].

REGULARIZATION

On the last three subsections of this part I will talk about three techniques to improve a model's performance. The first one is called regularization. It consists of a change in the cost function, a way to measure how well the model behaved. Let's

assume that a normal cost function is $C = E(W)$, where $E(W)$ is an error function and W the set of specific parameter values of a model. One possible problem that can arise is when the model overfits to the training data, getting extremely high values for W which would hinder the model when new data arise, as small data fluctuations would imply extreme model changes. This can be solved by adding a regularization term to the cost function, like this: $C = E(W) - \lambda R(W)$, where λ is the regularization parameter, which measures how much we want to penalize the model for large parameter values, and $R(W)$ is a function, normally increasing in value as $|W|$ increases. The two most common functions for $R(W)$ are:

- $\|W\|_1$: also called Lasso, this kind of regularization is recommended when feature selection is desired, as it can bring model coefficients down to zero, effectively filtering non-useful variables.
- $\|W\|_2$: also called Ridge Regression, it cannot force any coefficient to be zero, but it minimizes its impact on the final model. It is most recommended when collinearity is present.

There is also another method, proposed by Zou and Hastie [84], called elastic net, which combines both, getting the following cost function:

$$C = E(W) - \lambda(a \cdot \|W\|_1 + (1 - a) \cdot \|W\|_2) \quad (2.36)$$

where a is a balancing term that allows the user to decide the combination of Lasso and Ridge Regression they want to use. Thus, it allows to use the benefits of both methods while averting some of their caveats.

BOOSTING

Boosting is one way of combining multiple models in order to obtain a model better than the sum of its parts. Specifically, boosting is based on training each model in a serial way, optimizing each subsequent model to focus on the misclassified entries of the previous one. There are many boosting algorithms and methods (AdaBoost [85] being the most famous one) and the specific mathematics behind each

of them are not required for this explanation, the important thing is the intuition behind them: instead of combining a set of learners at the same level, boosting trains n learners one after another, making each one specifically good on the instances that the previous one was not good on. One caveat that Boosting shows is that the base learners utilized for it must be very simple.

RANDOM FORESTS

The last method I will talk about on this section is Random Forests. It is what is called an Ensemble Method: a combination of learners that, contrary to what happened in Boosting, are trained in parallel, without taking into account the other learners. Random Forests, specifically, is composed of a set of Decision Trees, which I already explained. In order to introduce variability in the growing of the trees, it applies randomness in two ways: first, it randomizes the set of variables each tree is grown with. And, second, it randomizes the set of observations it feeds to each tree. Thus, each tree is grown seeing a slightly different dataset. This, combined with a large enough number of trees, allows each tree to compensate the shortcomings of its peers. The exact hyperparameters of the model depend on the specific implementation used, but, normally, there are two: the number of trees to grow and how to split each tree, using the same criteria for Decision Trees [86].

2.3.2 EVOLUTIONARY COMPUTATION

I extract the following conclusions from my Observation Window setting issue review: the time problem is present in a variety of disciplines and there has not been a general solution proposed nor has the problem been studied profoundly. The papers that take it into account only test their method against several observation window values, with no assurance that the tested values will be optimal or even close to it. In order to fill this void, I propose a solution based on two ideas: the usage of a custom adapted Genetic Algorithm to optimize the observation window and the definition of multiple observation window schemes. My proposal does not

require human interaction nor previous knowledge of the system. Additionally, it automatically finds a good trade-off point between prediction performance and observation window size. I chose to develop our solution using a Genetic Algorithm because they are easy to adapt between problems, scale well and are robust. Of course, they are not without hindrances: they may not reach a global optimum and there is no clear process to tune them. I don't consider them serious problems, though, as the former problem can be solved by running them several times and the latter only affects computation time, not its performance. While I did not test them for this work, it would be feasible to use other optimization algorithms, too, and I would like to compare their performance on a future research line.

The usage of evolutionary algorithms in conjunction to a Machine Learning model is a well tested technique with a history of successful applications in several areas, such as electricity price prediction [87] and consumption [88][89], software reliability [90], financial [91] and stock data [92]. Every work I've found on this area follows the same idea: an evolutionary algorithm is used to optimize the parameters of complex Machine Learning models, such as Support Vector Machines [93] or Artificial Neural Networks [94]. These models have large parameter search spaces and their performance is highly dependent on their fine tuning. This task would, normally, require expertise on the domain area of the data or intensive brute force search, as the performance function is not something that can be easily optimized analytically. Evolutionary algorithms bridge this gap. Contrary to the trend, my proposal does not employ a Machine Learning algorithm that needs to be fine tuned, I do so because the complexity of our problem is on the data pre-processing: the evolutionary algorithm needs to be able to optimize tens of variables and, for each of those, its performance is interdependent on the values of the others. This makes the optimization problem quite complicated. More details on this can be found on Chapter 6. Considering this fact and the results obtained on Chapter 5, I chose to use Random Forests, as it is an algorithm that does not need to be extremely fine tuned to the problem for it to perform well enough. This allows the Genetic Algorithm to focus on optimizing the multiple window scheme I propose. There is also another difference between previous works on this dis-

cipline and mine: the application area. These techniques haven't been applied to the Network Failure Prediction problem and, as we studied earlier, my proposal is the first effort towards obtaining a general solution for it. Lastly, while a multiple window scheme has appeared in literature before [90], the optimization problem that paper solves is just finding the optimal value of a single number. My proposal deals with highly complex problems of, potentially, tens of interdependent variables. Such a problem has not been dealt with in related previous works.

My contribution can, thus, be summed up as a proposal of a general solution of the observation window size problem using a Genetic Algorithm and Random Forests. To do so, I adapt the basic Genetic Algorithm to the problem at hand with heavy customization of several parts of it, expand and study the multi-window scheme that appears in [25] to allow solutions to adapt to complex problems and validate my proposal using two real datasets, from two different areas, against other window configuration methods inspired by previous literature.

2.3.3 BIG DATA TECHNOLOGIES

I'm sure a contemporary reader will be totally aware of this, but, just in case: Big Data is the trending technology of recent years. And with no undeserved merits: it has allowed for the democratization of the computation and processing of massive amounts of data. This discipline has been defined as "... a set of techniques and technologies ... to uncover large hidden values from large datasets that are diverse, complex, and of a massive scale" [68]. This definition hints at several requirements that must be addressed when performing this kind of analysis. Normally, they are codified in what is known as the four V's of Big Data (which, as times goes by, have become more than four and incorporated other letters):

- Volume: this is the defining feature of Big Data problems, a quantity of data such that classical storage and computational techniques simply cannot be applied. This requires new algorithms and architectures, focused on scalability.
- Variety: another crucial characteristic of this new kind of analysis is the integration of different data sources, including structured and un-

structured data, in order to combine them to enrich the available information. Achieving this requires either a plethora of ad-hoc solutions, intense efforts of task automatizing or the definition of standards which everyone should follow.

- Velocity: elated to the volume problem, data must be analyzed at a sufficient pace to make the conclusions useful for the intended application.
- Value: the ulterior objective of these analyses, to extract hidden value from large amounts of data.

Let's remember that the last open problem I was tackling was the absence of practical implementations of Online Failure Prediction solutions on literature. My contribution is, thus, a software architecture implementation based on Big Data components and to prove its efficacy. So, regarding this section, my objective is to define and justify the software pieces I used to build it. Luckily, it is based on just two , which I will now detail.

APACHE SPARK

Apache Spark is a distributed computation system that allows the processing of large quantities of data swiftly. It provides high level APIs in Java, Scala, Python and R. It is supplemented by a wide array of high level tools that expand its capabilities, such as Spark SQL to work with structured data, MLlib and ML to perform Machine Learning tasks, GraphX to perform graph processing and Spark Streaming to process data streams.

It was originally developed at the University of California, Berkeley [95]. It has turned into the main open source software to perform distributed data processing [96] Its main innovation is a new implementation of MapReduce, the programming paradigm by which Hadoop [97] (the Big Data framework it practically superseded) functioned, that improves in several orders of magnitude Hadoop's one. It keeps, though, the almost linear scalability of MapReduce, but it improves its features by utilizing new data structures, as the RDD (Resilient Distributed

Datasets) and the Data Frame. The RDDs are the main data abstraction on Spark, allowing the user to perform a plethora of different processing techniques on large data volumes in a quick, easy, distributed and fault-tolerant way. They internally use the MapReduce paradigm and can be stored on the cache memory, which benefits algorithms that access the data several times or need a fast access to them. This ability is one of the main differences between Hadoop and Spark.

Regarding Machine Learning capabilities, there are two Machine Learning libraries, Spark MLlib [98] and Spark ML. The former is older and based on RDDs, whereas the latter is newer and uses Data Frames instead of RDDs, which improves the computational efficiency by utilizing a pipeline paradigm, opposed to the way RDDs work, which store new copies of the transformed data. These libraries have been shown to be the most recommendable when performing Big Data Machine Learning tasks [99] and, currently MLlib has absorbed the Data Frame based paradigm of Spark ML.

Lastly, Spark Streaming [100] allows Spark to perform fault tolerant, real-time processing to Big Data problems, by utilizing a microbatching paradigm, which groups the data flows as they arrive in small batches and then applies the desired set of operations to it. It has connectors to the most common data sources, as Apache Kafka, Apache Flume, Twitter, TCP sockets, etc.

APACHE KAFKA

Apache Kafka [101] is a register storage distributed service that allows to create pub-sub queues in order to connect Spark to any system that supports this kind of framework by establishing each data source as a publisher and different Spark modules (in my case) as consumers. In fact, I've already talked about this kind of architecture (using Kafka as a data connector and Spark as a processing platform) when I talked about reference [81].

R PROGRAMMING LANGUAGE

While it is not a Big Data technology *per se*, this section deals with the specific software I used to perform my analysis, so it is a good fit to explain a bit about this programming language. R is an open-sourced adaptation of S, a programming language created on the Bell laboratories in 1976 to substitute Fortran as an environment for statistical analysis. In 1993, Ross Ihaka and Robert Gentleman adapted it and open-sourced it, calling it R, and it was quickly adopted by the Statistics community. Currently R has greatly surpassed S in usage and popularity and has turned, along with Python, into the most used programming language for Data Analysis, Data Science and Machine Learning tasks. As such, its main usages are two: statistical analysis and modeling through Inferential Statistics and Machine Learning. Apart from these, it is also possible to carry out a myriad of activities on R: data visualization, automatic report generation, graph visualization... even webapp creation. It has an extremely healthy library of packages through the CRAN (Comprehensive R Archive Network).

2.4 CHAPTER SUMMARY

While on the Introduction I presented the problem in a broad way, here I focused a bit more and detailed the current situation of Failure Management on Computer Networks and Distributed Systems in general, showing how the current scientific literature is fragmented, mainly due to a lack of public benchmarks and, additionally, there is a disconnection between the problems that are usually solved in academic environments and industrial ones. I focused on two problems that need human expertise: the setting of the optimal Observation Window and the extraction of knowledge about the causes of failures. Furthermore, I also showed how there is no proposal of a realistic Online Failure Prediction software architecture to bring its concepts down to earth. I also presented a justification for the tools I will use in the different contributions, namely, Machine Learning methods, Genetic Algorithms and Big Data software pieces.

3

Objectives

As the literature review showed, there is a lot of work in certain areas of Failure Prediction, such as on Cloud Environments and High Performance Clusters, but surprisingly few works related to Computer Networks, and they are mostly fragmented and there are no current general methodologies that avoid needing human expertise. Furthermore, several decisions that need to be taken when performing Failure Prediction on realistic environments have not been tackled. These are the problem of information extraction, or Root Cause Analysis, the problem of deciding which time period to observe to create the Observation Window and, lastly, how to deploy a set of prediction models on a real system. Thus, the objectives of my thesis are aligned with solving these problems. I must also remark that, due to the methodology and problems dealt with, this thesis presents no hypothesis, only solution proposals and their validations.

Starting with the basic objective of **Solving the open problems in Distributed**

Systems Failure Management, I separate it into the following specific tasks:

1. Analyze a set of appropriate Machine Learning algorithms to find their behavior and performance in the distributed systems environments: this objective not only serves to characterize these algorithms in this area, but also supports the next chapters. By finding which algorithm performs correctly and is easy to use, I can decide objectively which one should the other contributions be based on.
2. Devise a general process to determine the optimal Observation Window for a prediction model.
3. Create a methodology to extract correlations and potential causes of failures from Online Failure Prediction models.
4. Propose a Machine Learning based Failure Prediction software architecture. The purpose of this objective is twofold: it narrows the gap between research and industry and serves as a feasibility demonstration of the concept of Online Failure Prediction.

3.1 SOLUTION REQUIREMENTS

While these objectives are still on a high level and need to be brought down to Earth on their corresponding chapters, there are several features of each solution that I can already identify as optimal or just necessary:

AUTONOMY

This is the most important feature and one that I've been repeating constantly. The main gripe and obstacle to overcome to achieve next-generation Failure Management is the need for human interaction and knowledge. Human interaction must be avoided because it imposes a reaction time ceiling: no person can react faster than a machine, more so as the system to be managed grows in size. In fact, at a

certain size, this does not become as much as a limitation as an impossibility. Huge systems cannot be managed by a single person, and the solution for this isn't optimal either: having a team of network managers adds another layer of interaction, which extends the time delay to react to observations. And having more than one expert makes even more obvious the second problem I talked about: the need for human knowledge. Obtaining expertise is hard, expensive, time-consuming and parts of it, the specific bits and quirks of a certain system will not be transferable. And if something happens to the expert, the knowledge is gone. This can be a catastrophe if there is no other expert available. It's clear to see that keeping humans out of the Management process as much as possible should be an objective to strive for. As such, the solutions I give to the problems should need as little human interaction as possible.

SCALABILITY

The main discourse behind the Autonomy requirement was the growing size of distributed systems as their importance in organizations and business increases. As such, we can expect them to keep growing and becoming more ubiquitous, so solutions related to them should be able to withstand this growth. Thus, my contributions should be able to minimize the computation and storage costs as they grow.

UNDERSTANDABILITY

While, ideally, no human should be needed to operate future management systems, that does not mean that the system should be a black box. As much as possible, how each part of it works and its outputs and methods should be easy to understand and alter. This is doubly important, given the relevance distributed systems have acquired: network operators should always be able to understand why some part of the system threw an alert or showed a significant time period or influence as relevant.

EASE OF USE

As it is unrealistic to expect network operators and managers to have any knowledge in Machine Learning algorithms or Data Science methods in general, each solution should be extensively documented, with its "knobs" (if any) clearly marked, to allow for tuning the system as desired.

RELIABILITY

A system's output should be as reliable as possible, this is, it shouldn't fail in its predictions. This can be achieved by imposing strict standards when building the system. Ideally, if the understandability and ease of use criteria are fulfilled, this is something that should be feasible to do by network operators.

UPDATABILITY

As systems grow and change, the management system should be able to adapt the inferred relations and prediction models to each new situation.

On next chapters I'll detail my contributions and the datasets I used to validate them.

4

Datasets Description

For the experiments that comprise my contributions I used two real datasets, one belonging to the realm of computer networks and provided to us by a Spanish bank, which I will call the N dataset, and one public dataset belonging to a High Performance Computation Laboratory in the USA, the Los Alamos National Laboratory. This dataset, which I will call LANL, is public [45]. I will now explore both of them in detail, as part of the data validation and exploration phases of a Data Science project. Fair warning: this section is much more conversational than a normal research report, as it will contain the exploration as it happened. I consider this organic way of presenting results the more appropriate one, as it allows the reader to follow the internal discourse I followed when performing the exploration.

4.1 COMPUTER NETWORK DATASET

This dataset contains network events codified by a network manager (a commercial product by Spectrum). The devices this manager monitors are part of the infrastructure giving service to a parts of the bank. While it has more than 10 variables, I will focus on the following ones:

- `Created_On`: the date and time the event was created on.
- `Name`: the name of the physical device in which the event appeared.
- `Model_Type_Name`: the network device category the device in which the event was created belongs to. There are switches, routers and virtual machines.
- `Event_Type`: a hexadecimal code indicating the type of the event that happened.
- `Event`: the message associated to the event. It expands and informs on the meaning of the `Event_Type` variable.
- `Severity`: a categorical variable indicating the importance of the event regarding potential harm to the network. It is divided in ""(blank), "minor", "major" and "critical".

The events contained in this dataset can vary from something innocuous as a configuration change on the TCP level to system-threatening situations such as losing contact through all protocols to the router, the virtual machines not running their specified softwares or the CPU of a device reaching certain emergency thresholds for an extended period of time. It contains data from the 16th of September, 2014 to the 9th of April, 2015, a period of 206 days. 21442 events happened on that period, on 21 different devices, averaging 9.82 events per day and device. Its event distribution by severity is shown on Table 4.1.1. On a first glance, it seems interesting to have a similar amount of Critical, Major and Minor events. This will,

of course, be explored along this section. I'll now study its distribution along time and by device.

Table 4.1.1: Number of event types and total events per severity on the Computer Network dataset.

Severity	Number of distinct event types	Number of events
Blank	73	16661
Minor	7	1614
Major	11	1539
Critical	4	1628

4.1.1 TEMPORAL DISTRIBUTION

First, we will see a histogram of events for the whole time period, where each bin represents 2 days (Fig. 4.1.1).

Figure 4.1.1: Event distribution of the Computer Network dataset.

This distribution is extremely interesting. We can observe two behavioral patterns and one extreme outlier: roughly on 2014 the even distribution is higher than on 2015, which even seems to follow a bit of periodicity for certain days. Obviously, the clearest observation is the huge spike occurred on the 24th of December of 2014. I will study it deeper later. For now, let's observe this histogram reducing the y axis limit to 600 events (Fig. 4.1.2).

Figure 4.1.2: Event distribution of the Computer Network dataset without the outlier.

Indeed, this plot shows more clearly how there seem to be two distinct behaviors before and after the 24th of December. In fact, now we can see that after a small period with no events before November 2014 (probably a shutdown) the average quantity of events diminishes. The contrary happens just before March 2015, where after several days of no events, the average quantity rises. There are, thus, four separate observable time periods: from the start to November 2014, from November 2014 to January 2015, from January 2015 to March 2015 and from this month to the end of the dataset. I also feel the need to remark that the semi-periodic spikes seem to happen all along the dataset. We will also study them further later. For now, let's try and find the number of events by month, to see if our observed behaviors are also shown there, by plotting Fig. 4.1.3.

Figure 4.1.3: Monthly event count of the Computer Network dataset.

While it may seem that the behavior here doesn't follow the four patterns I outlined, we must remember the time limits of the dataset: it starts on the 15th of September and it ends on the 9th of April. This means that, if they were following the same distribution, it would have almost 4000 events on September and around 3000 events on April, which seem to follow their closest months. And, in fact, if we took out the 2000 events of the 24th of December, that month would serve as a bridge between the large quantities of 2014 to the relatively inferior numbers of the beginning of 2015. The four periods I hypothesized seem to be supported by the data.

Let us now see if the average number of events per weekday is different. To do so, I will use the median, as it is a robust metric against outliers (and we've seen that there are several in our data) and show the number of events divided by weekday (Fig. 4.1.4).

Figure 4.1.4: Event distribution per weekday on the Computer Network dataset.

This boxplot shows what I was talking about before: Sundays seem to be very important in terms of raw quantity of events. It may be possible (and I'd say probable) that the bank performs maintenance operations on Sundays, thus the constant spike.

INCLUDING SEVERITY

To further understand how this system behaved, I am now going to introduce the Severity variable to the analysis. Recreating the monthly chart taking Severity into account we get Fig. 4.1.5. Now these two plots confirm what I've been talking about: until January, the environment was much more unstable, shown by the sheer number of events and the percentage of them that are critical (around 20% for the first three months). It got fixed for January, but the errors kept creeping up, shown by the quantity of Major events rising steadily as time went by. Also, it seems that the event peak of the 24th of December had an extreme amount of Blank events, suggesting that, maybe, it wasn't such a catastrophe as it looked before. We will now see the weekday distribution of events, separated again by their severity, on Fig. 4.1.6.

Figure 4.1.5: Event distribution on the Computer Network dataset divided by Severity.

Figure 4.1.6: Event distribution on the Computer Network dataset divided by Severity and weekday.

While the plot is informative for the Blank events (suggesting, again, that Sundays are for maintenance operations), their range squashes the other plots. Fig. 4.1.7 shows the same plot without Blank events. Here the "Sunday effect" seems to be apparent (and not very pronounced) for Major events. Overall the weekday does

not seem to have much of an effect on these distributions. This could be further checked using an Analysis of Variance (ANOVA), but having seven levels would mean to incur in a lot of error if I tried to check the pairwise differences. Thus, I won't explore this issue in more detail.

Figure 4.1.7: Event distribution on the Computer Network dataset divided by Severity and weekday without Blank events.

4.1.2 DEVICE DISTRIBUTION

As I previously did, exploring the dataset on the time dimension, I am now going to study it focusing on its devices. Luckily, when we obtained this dataset we had help from the network operators, who helped us label each device name with their correct device type (while there is a `Model_Type_Name` variable, it is not highly informative). First, let's see what is the actual device distribution: there are 21 devices, of which there is only one router, six switches and 14 virtual machines. This looks, then, like a small part of a service network. Sadly, we have no description of the network topology (though Contribution 3 deals precisely with this issue). As a first analysis, I'll detail the dataset's event severity distribution based on the device type (Fig. 4.1.8).

Figure 4.1.8: Event distribution of the Computer Network dataset by device type and Severity.

While the Virtual Machines hold the first position in the largest number of events, something we could expect based on their numbers, the router (a single device) suffers more events than six switches together. One could argue that this is based on the high number of Blank events that may be due to configuration changes, but there's also a significant number of Minor and Major events. I should also note that the virtual machines hold the majority of the Critical events, by large. On the next subsection I will analyze the events on their own, where we'll see which type of events they suffer. For now, let's subdivide the device category into each single device, to see if this throws some light on the distribution issue. This is shown on Fig. 4.1.9.

Figure 4.1.9: Event distribution of the Computer Network dataset by device.

This plot is highly informative regarding the network activity. The router appears as the busiest point of the network and a subgroup of virtual machines has a lot of activity, as well as the A group of the switches. If divide the events are now divided based on their severity (4.1.10) it is discernible now that Virtual Machines 12, 8, 10 and 14 are the ones where most critical events happen. Also, it is now observable that every switch has a small, but not negligible, number of critical events happened on it.

Figure 4.1.10: Event distribution of the Computer Network dataset by device.

4.1.3 CRITICAL EVENT ANALYSIS

Lastly, to finish my exploration of this dataset, I will perform a detailed analysis of each Critical event type. There are four distinct types:

- Critical Event 1: this event indicates that a device and its contained blades have stopped responding to polls and requests. Additional metrics imply a chassis fault.
- Critical Event 2: this event is a precursor of the Critical Event 1. It signals that a device has stopped responding to polls.
- Critical Event 3: a device is suffering a resource threshold violation. It may be CPU, memory usage, disk usage, or some other indicator.
- Critical Event 4: a Java executable is not running enough instances. This process is monitored by a certain virtual machine. It seems to indicate that the software processes that this network supports have stopped working.

There are, thus, three different domain areas for the critical events: hardware, network and application. I will now analyze the temporal and device distribution of each of them.

CRITICAL EVENT 1

As we saw on its description, this event signals a complete hardware fault. Thus, it is extremely important, as it can render a whole network useless depending on which device is affected. In a similar way to what I did with the whole dataset, I will study its appearances based both on the moment they appear and where they do. Thus, regarding time, the monthly distribution of this event is shown on Fig. 4.1.11. And it is quite revealing. The ample majority (except just two instances) of this event happened on December. Could they be related to the huge spike at the end of it?

Figure 4.1.11: Critical Event 1 monthly distribution.

Fig. 4.1.12 answers the question I posed on the previous paragraph. Indeed it is. This finding supports the fact that something important happened on the network on the 24th. So, overall, we have three isolated events and forty nine events related to the 24th spike.

Figure 4.1.12: Critical Event 1 distribution in December.

Dividing the plot by device results in Fig. 4.1.13. The forecast I made is slightly altered now, due to the fact that every switch was affected by the event at the same time. It now looks like a massive network change... or a major catastrophe. Without further information, I cannot distinguish between the two of them. The fact that the two isolated events affect the router are also consistent and support the fact that this event can bring the whole network down.

Figure 4.1.13: Critical Event 1 monthly distribution separated by device.

CRITICAL EVENT 2

This event is the precursor of the one I just analyzed. Knowing this, I would expect it to happen in December, January and April, at least. Further instances of it would signal moments when the affected device did not completely go down. As usual, let's first see its monthly distribution (Fig. 4.1.14).

Figure 4.1.14: Critical Event 2 monthly distribution.

While most of the results are as I expected, there is one thing that isn't: there is no precursor event for the Critical Event 1 of April! This is quite important, as it may mean that either the underlying cause for the Critical Event 1 can happen suddenly without warning or that the monitoring system is not as precise as it should

be. The device distribution (Fig. 4.1.15) should help us understand more about this event.

Figure 4.1.15: Critical Event 2 monthly distribution separated by device type.

Well, this plot has completely subverted my expectations. The router only suffers this event once, whereas it had suffered its (supposed) successor twice. And the virtual machines suffer this event quite often... but they don't suffer Critical Event 1! While this behavior is very different from what I expected, it is a valuable finding (and, in fact, could serve to improve the monitoring system if this was the objective of this research).

CRITICAL EVENT 3

Whereas the previous events dealt with the network level and reachability, this is a purely hardware-related event: the usage of a certain resource over a specified threshold. Fig. 4.1.16 shows when and how much it happened. Very localized on the first months of the dataset: consistent with the wilder behavior we observed on the beginning of this exploration. Now, adding the device (Fig. 4.1.17) we find that, as they are the ones dealing with varying workloads, it is consistent that it is a subset of the virtual machines the one that has resource usage problems. On a positive note, it is only on two of them (and the overwhelming majority is on just one of them), which suggests a specific problem, not a systemic one or a flaw in the whole system design.

Figure 4.1.16: Critical Event 3 monthly distribution.

Figure 4.1.17: Critical Event 3 monthly distribution separated by device.

CRITICAL EVENT 4

This last critical event is the only one software related: it controls that the processes this network provides the foundations for are actually running. Fig. 4.1.18 contains its temporal distribution. And, at last, we find the main offender for most of the critical behavior of the first months of the dataset. Based on its description, it should only appear on virtual machines. But I still don't know if it's focused on a certain virtual machine or if it's a generalized problem. Fig. 4.1.19 attempts to address this issue. There are several facts to extract from this plot: it seems that four virtual machines (out of fourteen) are the ones having these problems. And they seem to continue, unattended, from September to October (let's recall that the dataset starts at the middle of September, so the actual number of events in

both months is probably very close). Then, for November, it looks like they were able to (mostly) correct all the problems for virtual machines 10 and 14, whereas they were the same for the other two. They were all corrected by the end of the month, though, as they don't appear again for the rest of the dataset.

Figure 4.1.18: Critical Event 4 monthly distribution.

Figure 4.1.19: Critical Event 4 monthly distribution separated by device.

4.1.4 EXPLORATION SUMMARY

I have explored my Computer Network dataset in several dimensions: time, the different network devices and their categories and I've made a special focus on the four types of critical events. The most important facts I've learned from these analyses are:

- There are four different behaviors, marked by declines and rises of the total number of monthly events.
- The network was much more faulty the first three months (thus, most of the critical events are focused on these), culminating on a network-changing event on the 24th of December, which rendered every switch unreachable.
- The virtual machines hold most of the critical events of the system, mainly, excessive resource consumption and missing software processes.
- While two of the critical events seemed to be closely related, they actually don't seem to be so, something that should be taken into account by network operators and that I will check in the next chapters.

4.2 LANL DATASET

This dataset contains events and errors, modeled after a "remedy" database, that happened at the Los Alamos National Laboratory's High Performance Computer clusters, composed of 23 systems, with a time range of 3765 days (10 years and 115 days), starting on the 19th of May 1995 and ending on the 9th of September of 2005. The dataset has 23739 rows, which shows a much lower event density than our previous dataset. Out of the twenty-six variables this dataset has, these are the ones I will use for further analysis:

- System: the affected HPC system. It is a numeric variable that goes from 2 to 24, though system 7.
- Nodes: number of nodes in the system.
- Nodenum: which node was affected by the error.
- Node Install: date when the node was installed.
- Node prod: date when the node was placed in production.

- Node decom: date when the node was taken out.
- Problem.Started: the exact date when the problem appeared.
- Error type variables: there are six columns which, together, indicate the type and subtype of the event that happened. They are Facilities, Hardware, Human Error, Network, Undetermined and Software. In order to facilitate our analysis, we will transform these six variables into two: Error_Type, containing just the error category, and Error_Subtype, containing the error subcategory.

While the dataset is much richer than this, we will see that my experiments only need the variables I've extracted. I must also remark that there is no severity measure here, so I won't be able to distinguish which problems are more important than others. Additionally, there were 453 events with no specified event cause, so I set their error type to "Missing". I will follow a similar structure on the exploration of this dataset to the one we followed for the previous one, with two main differences: the study of severity will be changed by a study of the effect of the number of nodes; and the critical event analysis will be swapped with a study of all causes and their subtypes.

4.2.1 TEMPORAL AND SPATIAL ANALYSIS

As I have already done before, I will start analyzing how the events are spread over time. We've already seen that the event density will be much lower than on the other ones, thus, 4.2.1 shows its temporal histogram, where each bin is approximately a month. What we are observing here may be, indeed, a common pattern for distributed systems: an initial behavior followed by a sudden spike (that may indicate certain changes or optimizations on the system) followed by an upwards trend of increasing events due to system aging. It is quite interesting that we've observed this for both of the datasets (and let's not forget that they come from totally different backgrounds and represent different systems).

Figure 4.2.1: Event distribution of the LANL dataset.

I will now add the type of error to the previous plot. Additionally, we are going to observe yearly trends instead of monthly to make it easier to analyze (4.2.2).

Figure 4.2.2: Event distribution of the LANL dataset separated by error type.

There are several useful insights we can get from these charts: overall, hardware problems are the most abundant, specially on latter years. In fact, without consid-

ering 1995 and 1996 (as events then were extremely low), hardware failures are the most common every year except on 1999 and 2006. Software problems are the second most frequent category, with a rise on 1999 and a representing a steady proportion of the total number of events each year afterwards. It is also very interesting to see how the Missing events (the ones that had no specified cause), represent a considerate proportion of the events of the first years and then diminishes greatly. Undetermined events, though, for which there was no known cause, remain at a steady proportion yearly. This isn't something I'd expect a priori, considering that it would be expected for the knowledge base to grow over the years. Studying the underlying cause for this process would be a fascinating effort, as they are almost always the third most common error category.

Conversely, I will now study the temporal evolution of the number of events per system (4.2.3).

Figure 4.2.3: Event distribution of the LANL dataset separated by system.

It seems that system 2 is the one with most errors, being overrun by systems 18,

19 and 20 as time goes by. This indicates that most failures seem to be centered on a small amount of the systems. But there is one variable that we are not taking into account: the number of nodes on each system may quite easily skew the number of events in favor of bigger systems. To avoid this, Fig. 4.2.4 shows the ratio of errors over number of nodes per system. There are seven systems (15, 23, 7, 2, 24, 16 and 22) with a ratio value over 50, though most of them don't have a large number of events. The ones that combine "error-proneness" with a high error frequency are systems 2 and 16. But there is one more thing we are not considering that may as well impact the data: the time a system was online. Unfortunately, we only have the deployment and decommision times for each node in each system.

Figure 4.2.4: Ratio of errors per number of nodes for each system of the LANL dataset.

What I can do, though, is to infer lower and upper bounds for each system's time using these. Doing so, we get the time distribution found on Fig. 4.2.5. On it, systems are sorted following their event ratio found on the previous one. We can see that the most error-prone are, in fact, the ones that were deployed on the first years and that there was a big deployment of new nodes on 2004. Using this time value, I can now correct the ratio, where instead of using the number of events divided by the number of nodes, I further divide that value by the number of days the system was online. This corrects the effect of having been deployed for a different time. For the sake of clarity, I'll normalize this new ratio between 0 and 1. Results are shown on Fig. 4.2.6.

Figure 4.2.5: Time amount each system of the LANL dataset was deployed, measured in days.

Figure 4.2.6: Normalized ratio of errors per number of nodes and time deployed for each system of the LANL dataset.

Comparing to the similar ratio scatterplot I previously showed, there are two systems that have taken a leap upwards from the trend we observed: systems 15 and 7. The big leap for system 15 is based on the combination that it was deployed for a very short time, comparatively, and it only has one node on it. That is the cause for system 7's slight increase: although it was part of the first wave of deployments, it was decommissioned much earlier than its peers and it is also composed of a single

node. While these two systems wouldn't raise any suspicion in simpler analysis, taking all the variables into account, we can see that they are certainly different from their peers.

While I've analyzed the dataset on a system level, we can go deeper, as we know, for each failure, which was the affected node of which system. There is one additional source of variability, though: how much time a node has been deployed. Luckily, this time I don't need to infer it, it is a supplied value. This lets me derive an index, similar to the previous one, but at a node level. As there are 3576 different combinations of system and node, we will first check the general distribution and then focus on the most error-prone ones. This analysis is done on Fig. 4.2.7.

Figure 4.2.7: Normalized ratio of errors per time deployed for each node of the LANL dataset.

The plot is clear: it presents a typical Pareto distribution, where most nodes have a very small ratio and there is a clear change at a certain point where the ratio spikes, around 0.05. In fact, out of the 3576 nodes, 3446 have a ratio lower than this value, a 96.36% of the data. This would allow a network operator to focus only on those points. Filtering most of the values, the system distribution of the top 30 error-prone nodes is contained on Table 4.2.1.

Table 4.2.1: System distribution of the top 30 error-prone nodes on the LANL dataset

System	2	16	23	7	15	18	19	20	21	22
Frequency	11	9	3	1	1	1	1	1	1	1

These results support the conclusions I extracted on a system-wide level: systems 2, 16 and 23 also appeared on the system list. This list would also be useful to find certain nodes of otherwise healthy systems which malfunctioned. As an example, Table 4.2.2 shows the 10 nodes with the highest ratio overall.

Table 4.2.2: Top 10 nodes with the highest normalized error number per time deployed ratio on the LANL dataset.

Node	16-1	16-13	15-0	2-1	16-2	2-21	16-3	19-1024	23-2	23-1
Ratio	1	0.631	0.623	0.603	0.532	0.483	0.432	0.395	0.378	0.340

The only system here that I had not identified as faulty is system 19. The table gives us the reason: it is node 1024, while that node may not be working correctly, its effect was diluted between the other hundreds of nodes. This analysis could be expanded to continue finding single nodes, but for the scope of our exploration, this is already enough.

4.2.2 ERROR CAUSE ANALYSIS

As a last and different point of view, I'll analyze the different error types and sub-types of the dataset. First, Table 4.2.3 shows the apparition frequency of each event type.

Table 4.2.3: Event Type percentage distribution on the LANL dataset

Event Type	Hardware	Software	Undetermined	Missing	Network	Facilities	Human error
Percentage of apparitions	60.52%	22.45%	11.18%	1.92%	1.78%	1.51%	0.63%

We've already seen this result: hardware and software errors are the most frequent ones but the undetermined are also a substantiate amount which should be studied. Fig. 4.2.4 addresses if the distribution per system (focusing on the five most error prone systems) is similar.

Table 4.2.4: Event Type percentage distribution on the five most error prone systems of the LANL dataset.

System	Hardware	Software	Undetermined	Missing	Network	Facilities	Human error
2	54.15%	26.28%	12.67%	0%	4.04%	2.32%	0.53%
7	19.38%	34.11%	7.75%	32.56%	0.78%	5.43%	0%
16	36.60%	29.74%	18.06%	12.09%	0.90%	1.49%	1.12%
22	45.53%	14.23%	8.54%	19.92%	1.22%	9.76%	0.81%
24	43.23%	10.32%	18.71%	19.35%	0.65%	7.74%	0%

Overall, i infer that the distributions move around the general percentage for most cases, being the Missing errors a special case, as they represent a much higher number for these systems than for the general case. This may be due to the fact that the systems we've looked at were deployed on the environment's infancy. To get more insight on it I need to focus on each type on its own. I will do so for the three most common categories.

HARDWARE ERRORS

This type represents the majority of the errors on the dataset, being 14291 out of 23613, this is, a 60.5%. Its distribution over time per system is shown on Fig. 4.2.8. The trend here is similar to the one we observed for the whole dataset: systems 2 and 16 get the greatest percentage and absolute number for the early years, gradually disappearing in favor of systems 18, 19 and 20 in the last years.

This error category has 98 different subtypes. 50% of them happen 9 times or less, while 75% percent of them do so 38 times or less. CPU and memory errors are, by far, the most common ones, happening more than an order of magnitude more often than the other ones. Nevertheless, there is a great amount of variability on this error category, as there are a lot of different subtypes that happen non-negligible times.

Figure 4.2.8: Number of hardware events over time divided per system on the LANL dataset.

SOFTWARE ERRORS

The distribution landscape of this type (Fig. 4.2.9) follows the same patterns we’ve observed both in the general distribution and the hardware one (except for a great spike on system 2 in 1998). Regarding how this category is further divided, it does so in 30 subtypes. In a different way to the previous category, Hardware, they seem to be much more evenly distributed, due to the fact that the median value of each subtype’s apparitions is 57.5. On the other hand, it also presents a subgroup of really frequent subtypes. As a sort of ‘catch-all’, the most common subtype is “Other software”. This could be expected, seeing that this category probably includes everything not directly classified. But the second and third categories, DST and OS are very close to it. While I searched but could not find what DST meant, it is sensible to understand how the other categories are important: they deal with the basics of the nodes, its OS and the file system.

Figure 4.2.9: Number of software events over time divided per system on the LANL dataset.

UNDETERMINED ERRORS

This is a special category, as it deals with errors for which there was no known cause. We could expect, probably, to find several underlying temporal and spatial distributions forming this category, as, as with happened with the "Other Software" subtype, this error is a "catch-all" for any non categorized event. Its spatiotemporal distribution is on Fig. 4.2.10. Indeed, the error distribution is quite different now, specially on the second wave of system deployments: this category is more spread all over the new systems, instead of focused on a certain subset of them. Being a special category, its subtypes are also different: there are only three, "Unresolvable", "Undetermined" or "Security". Fig. 4.2.11 shows the data from Fig. 4.2.10 further subdivided by the error type. Supporting the hypothesis we presented, the three of them present totally different distributions: the unresolvable events seem to be a thing of the first wave, whereas the undetermined ones belong mainly to the second one. Lastly, the security events seem to be a minority, small anecdotes over time.

Figure 4.2.10: Number of undetermined events over time divided per system on the LANL dataset.

Figure 4.2.11: Number of undetermined events over time divided per system and type of event on the LANL dataset.

4.2.3 EXPLORATION SUMMARY

On this section I explored the Los Alamos National Laboratory High Performance Computing Cluster dataset. I did so by time and by system, showing how the cluster behaved completely different between "eras" and the influence of each specific system and node on the error distribution. This knowledge should be leveraged when creating prediction models in order to optimize their performance. Additionally, I did the same kind of analysis for the three most common types of errors. This kind of deep analysis can shed light on the features of a dataset and is a crucial part of the Data Science process. Additionally, I've introduced a slightly more complex way of exploring a system, instead of just summarizing its raw variables: the usage of ratios can serve to unmask certain effects that were being hidden by a system size or its deployment time, for example.

Now that I've described and explored the datasets I'll use, the following chapters present the different contributions of my thesis, validated using these data.

4.3 DATA PREPROCESSING

We studied on the Online Failure Prediction State of the Art section that, in order to study a dataset to predict the future, some kind of preprocessing must be done to the serial data in order to turn it to something a Machine Learning algorithm can understand and process. For both of my scenarios, I chose the following transformation: being $X = \{x_1, x_2, x_3, \dots, x_n\}$ a sequence of n events, each of them characterized by a certain event type $K = \{k_1, \dots, k_l\}$ and happening on a certain time $T = \{t_1, \dots, t_n\}$, there are two processes to follow. The first one is the creation of the Observation Windows, $W_o = \{W_{o,1}, \dots, W_{o,n}\}$. These are created as follows: selecting a certain observation window size, S_{wo} , and for each t_m , a numeric vector of size l is created containing the number of times an event of each type happened between $t_m - L$ and $t_m - L - S_{wo}$, being L a specified lead time. Thus, n Observation Windows are obtained, one for each event that happened on the system. Lastly, a dummy variable is added to each of them indicating whether no events happened

on a window in order to avoid having blank rows on the dataset. Then, the Prediction Windows are created. Whereas the Observation Windows are common for the whole dataset, no matter which event we want to predict, the Prediction Windows are specific for a certain event type, k_s . The creation of a Prediction Window vector for a certain event type, Y_s , is defined as:

$$Y_s = \{\delta_{s,1}, \delta_{s,2}, \dots, \delta_{s,n}\}$$

where

$$\delta_{s,i} = \begin{cases} 1, & \text{if } k_i = k_s \\ 0, & \text{if } k_i \neq k_s \end{cases}$$

Thus, each Prediction Window vector is binary, making my prediction tasks a supervised classification task.

5

Contribution 1: Machine Learning Models Comparison

We've now reached the first contribution of this thesis. The objective of this one was twofold: first, I wanted to find a base Machine Learning algorithm that I could use consistently on the other proposals. For this, I needed to check their raw performance and their parameters' behavior. Apart from that, this contribution also seeks to bring up the Failure Prediction research area up to the same level of other similar areas, such as Anomaly Detection, which sport surveys that compare and benchmark different models. As happened on Chapter 4, this contribution contains, mainly, the exploration of the behavior of different algorithms performance in terms of their parameters' values. As such, it is also quite conversational on the exploration part. This chapter's contents started as papers [11] and [12].

5.1 EXPERIMENTS DESCRIPTION

I compared an array of different Machine Learning models on the task of predicting the appearance of five events of each of the datasets I've just described. These were chosen in terms of their frequency of appearance and importance. Every model was validated following the same procedure: 10 folds cross validation, repeated three times in order to decrease the potential variability of the results. Due to the fact that, by the nature of our data, there is a huge imbalance between our target event samples and the negative samples, I must carefully choose how to evaluate a model's performance. The most simple metric, for instance, would not be useful: Accuracy, just calculating the percentage of samples you correctly classify, is not an effective metric for such cases. I will use Cohen's Kappa, defined as the ratio of interagreement between our models and the real data with the formula:

$$k = \frac{Pr(a) - Pr(e)}{1 - Pr(e)}$$

These values are calculated based on the confusion matrix, which I will show again to prevent moving back and forth on sections:

		[h]	
		True	False
Predicted Value	Real Value	True	False
	True	True Positive (TP)	False Positive (FP)
	False	False Negative (FN)	True Negative (TN)

Using it, Cohen's Kappa values are:

- $Pr(a)$: agreement percentage between the model and the real data: $Pr(a) = \frac{TP+TN}{TP+TN+FP+FN}$.
- $Pr(e)$: expected agreement level if the model was outputting random predictions. This is the value that helps taking into account the class imbalance. Its formula is: $Pr(e) = \frac{(TP+FN)*(TP+FP)+(FP+TN)*(FN+TN)}{(TP+FN+FP+TN)^2}$

Thus, this value can be taken, intuitively, as the percentage of "useful" information that the model is predicting. All data were preprocessed following the windowing process I described on the State of the Art chapter. I used a 5-minute observation window for both datasets. As every model has certain parameters to tune and some of them require extensive expertise, I avoided this by performing a random search over the parameter space. The number of different combinations of parameters I tried was 100, as an arbitrarily large enough number. All of these experiments were done using the 'R' programming language and statistical environment, powered by the 'caret' library, running on an HP ProLiant SL250s Gen8 with the following specifications: two Intel Xeon e52630v2 2.6GHz (6 cores) and 32GB of RAM. The models I chose to use were based on the most common methods applied in literature, and my own experience in applying Machine Learning to OFP problems. As all of them were explained in detail previously, here I will just give an account of which parameters they have and the exact 'R' library I used to train them (Table 5.1.1).

Table 5.1.1: Tested Machine Learning algorithms and their software implementation.

Model Name	R Library	Model Parameters
Random Forest	ranger	<ul style="list-style-type: none"> • mtry: number of randomly selected predictors. • splitrule: how to split the dataset when growing trees. • min.node.size: minimal tree node size.
Support Vector Machine	e1071	<ul style="list-style-type: none"> • sigma: width of the gaussian kernel. • C: the cost, a penalty factor.
Cost-Sensitive CART	rpart	<ul style="list-style-type: none"> • cp: complexity parameter. • Cost: Cost.
Elastic Net Logistic Regression	glmnet	<ul style="list-style-type: none"> • alpha: mixing percentage. • lambda: regularization parameter.
Boosted Logistic Regression	caTools	<ul style="list-style-type: none"> • niter: boosting iterations.

And these were the events I chose to model. For the Networking dataset:

- **Event N1:** (Critical Event 4 on VM8) a software process is missing on a certain node.

- **Event N2:** (Critical Event 3 on VM6) a CPU usage threshold has been surpassed.
- **Event N3:** (Major Event 5 on Router) a communication link is down.
- **Event N4:** (Major Even 2 on Router) a device is not responding to primary management requests.
- **Event N5:** (Major Event on Switch A2) contact with a chassis has been lost.

And for the LANL dataset:

- **Event L1:** a Hardware error (CPU subtype) on system 18.
- **Event L2:** a Hardware error (Memory Dimm subtype) on system 2.
- **Event L3:** a Software error (DST subtype) on system 2.
- **Event L4:** an Undetermined error (Unresolvable subtype) on system 16.
- **Event L5:** a Missing type error on system 16.

On the following subsections I will first study the effect of each algorithm's parameters on its performance and, after that, we'll extract a summary for each dataset's results in terms of prediction accuracy and computation time.

5.2 MODEL RESULTS

This section will go over the five different algorithms I tested, analyzing their parameters and their effects.

5.2.1 RANDOM FORESTS

As the Random Forests implementation I are using has three parameters to tune, there are three dimensions to explore, as well as 10 different distributions over them. The effect of the first parameter 'mtry', is shown on Fig. 5.2.1.

Figure 5.2.1: Kappa for each tested event based on the number of trees grown.

As expected, the number of trees that are grown on a Random Forests model is just a threshold to overcome: after the number of trees reaches a certain number (around 200 or 300 depending on the event) the performance plateaus. This value seems to have more influence when the data are harder to predict (events N1 and N2). Conversely, for events N3, N4 and N5, only extremely low numbers of trees harm the performance.

The second parameter, 'splitrule' is a category with only two values. Adding them to the previous plot, we get Fig. 5.2.2.

Table 5.2.1: Tukey range test results on the effect of the split rule on each event's model's performance.

N ₁	N ₂	N ₃	N ₄	N ₅	L ₁	L ₂	L ₃	L ₄	L ₅
0.038	0.0373	0	0	0.091	0.035	0.086	0	0.011	0

Figure 5.2.2: Kappa for each tested event based on the number of trees grown and the split rule.

The plot on itself doesn't yield much information, so I will run a Tukey range test on an Analysis of Variance separated by event to check if the split rule is causing a significant difference. Its results are summarized on Table 5.2.1.

The table shows the average difference between rules, where a positive difference indicates that the Gini rule was better and an 'NA' indicates no significant difference. It is clear that the Gini rule is the best of the two (though not by a large difference).

Lastly, I have to analyze the 'min.node.size' parameter. Its effect on its own is shown on Fig. 5.2.3.

Table 5.2.2: Linear model coefficients for the effect of the RF parameters on the model performance.

Parameter	Coefficient	$Pr(> t)$
min.node.size	-0.054	0
mtry	0.14	0
splitrule(gini)	0.046	0.000912

Figure 5.2.3: Kappa for each tested event based on the the minimum node size.

While it does seem to have a bit of influence on a small set of the events, I cannot guarantee that it is caused by this parameter. In order to analyze the effect of the three parameters at the same time, as well as their interactions, I analyzed the coefficients of a linear model of the Kappa score in function of them (results on Table 5.2.2).

The coefficient for the min.node.size parameter is negative, which seems to indicate that forcing large nodes on the trees hurts the performance. Overall, I cannot say that the parameters (except from some fringe cases of mtry) influence the model's performance in great avail. This is expected, and, in fact, a desired feature of Random Forests: consistently good performance without the need of extensive tuning.

5.2.2 COST SENSITIVE CART

The second tree-based learner I used on the data, this one trains just a single tree but improves the cost function of the algorithm by taking into account class proportions. It has two different parameters, cp (a complexity parameter) and $cost$, the class weight. The complexity parameter relation with the performance values is shown on Fig. 5.2.4. While there are several different behaviors, overall, performance seems to decrease as the complexity value (that ranges from 0 to 0.1) increases. On the other hand, the behavior for the cost parameter can be seen on Fig. 5.2.5. For this parameter we observe decreasing performance as it increases. Nevertheless, the best options don't seem to be always a cost of 0, so there seems to be some usefulness in including the cost.

Figure 5.2.4: Kappa for each tested event in terms of the complexity parameter.

Figure 5.2.5: Kappa for each tested event in terms of the cost parameter.

To check the assumption of having value in having a cost different than 0, I show on Table refkappa-cart the best parameter combinations for each event and their model's performance. Its results confirm that, for most events, a cost different than 0 is the best option, suggesting that the class imbalance was addressed by the cost compensation.

Table 5.2.3: Best parameter combinations and Kappa values for the Cost-Sensitive CART.

Event	Cost	cp	Average Kappa	Kappa Standard Deviation
N1	1.249	$2.53e-5$	0.111	0.264
N2	1.249	$2.53e-5$	0.423	0.154
N3	1.717	$3.00e-7$	0.899	0.348
N4	1.249	$2.53e-5$	0.831	0.028
N5	1.717	$3.00e-7$	0.939	0.054
L1	1.965	$8.52e-5$	0.003	0.017
L2	7.838	0	0.096	0.078
L3	1.965	$8.52e-5$	0.052	0.023
L4	1.965	$8.52e-5$	0.004	0.014
L5	1.965	$8.52e-5$	0.296	0.104

5.2.3 SUPPORT VECTOR MACHINES

The third model I analyzed is Support Vector Machines. Contrary to Random Forests, this model is famous for its need for fine tuning in order to achieve high prediction results. We'll check if this is true for the chosen events.

The implementation of the SVM I am using has two different parameters, sigma and C, a cost value. We already studied their meaning on the State of the Art section, so I'll go ahead and study their influence on the models' performance. Repeating the same process of analyzing a linear model and analyzing its coefficients, we get the results on Table 5.2.4.

Table 5.2.4: Linear model coefficients for the effect of the SVM parameters on the model performance.

Parameter	Coefficient	$Pr(> t)$
sigma	0.1021	0.025
C	0.0004	0
Interaction term	0.0007	0.071

Both parameters seem to affect the model's performance, especially the cost. As another way to analyze their effect, I'll plot the standardized performance of each model as a function of them (Fig. 5.2.6).

Figure 5.2.6: Standardized Kappa of the SVM models in terms of their parameters. Green color indicating that the model obtained a Kappa higher than 2 times the standard deviation of the set.

Based on the plot, it seems that low values of C and σ benefit the model, higher values of σ allowing lower values of C . Lastly, the best parameter combinations for each model are shown on Table 5.2.5.

Table 5.2.5: Best parameter combinations and Kappa values for the SVM.

Event	sigma	C	Average Kappa	Kappa Standard Deviation
N1	0.016	854.694	0.589	0.417
N2	0.062	53.502	0.514	0.146
N3	0.001	299.176	0.909	0.030
N4	0.036	210.404	0.873	0.027
N5	0.002	137.575	0.967	0.031
L1	0.098	83.343	0.234	0.095
L2	0.091	126.824	0.365	0.056
L3	0.098	83.343	0.410	0.021
L4	0.157	46.659	0.316	0.057
L5	0.023	430.267	0.594	0.104

While the performance values are extremely varied, it seems that, overall, the best results are obtained by having small values of σ (in the order of 10^{-3}) and high values of C .

5.2.4 ELASTIC NET LOGISTIC REGRESSION

This model has two parameters, α , which combines the two regularization types, and λ , the regularization coefficient. As I did for the rest of the models, I will study both parameters on their own first. Fig. 5.2.7 shows the effect of α on the model's obtained Kappa, whereas Fig. 5.2.8 does the same for λ .

Figure 5.2.7: Kappa for each tested event in terms of alpha on the Elastic Net LR models.

Figure 5.2.8: Kappa for each tested event in terms of lambda on the Elastic Net LR models.

It's patently clear that higher values of lambda don't produce good results. Thus, I will analyze the effect of alpha for low values of lambda on Fig. 5.2.9 (less than 0.15 but higher than zero). I cannot distinguish any kind of relevant pattern on the effect alpha on the model's performance.

Table 5.2.6: Linear model coefficients for the effect of the Elastic Net LR parameters on the model performance.

Parameter	Coefficient	$Pr(> t)$
alpha	-0.031	0.209
lambda	-0.035	0
Interaction term	0.004	0.71

Figure 5.2.9: Kappa for each tested event in terms of alpha taking only values such that $0 < \lambda \leq 0.15$.

Lastly, I'll also analyze the effects of the parameters on a linear regression of the performance value (Table 5.2.6).

These results seem to support the conclusion that the regularization did not increase (or even affect) the performance of the model. This suggests that a normal logistic regression would perform better than this model. To assert this, we move to the next algorithm I evaluated, Boosted Logistic Regression.

5.2.5 BOOSTED LOGISTIC REGRESSION

Boosted Logistic Regression only needs one parameter to be tuned: the number of boosting iterations, or 'nIter'. I expect this parameter to act in a similar way to the number of trees on Random Forests: as a threshold, if it is large enough the model should reach a performance plateau. Being just one parameter, a single plot (Fig. 5.2.10) will be enough to study the search. While my threshold assumption seems to hold true, for most models the curve is not too pronounced. There are others, though (such as event N₁) where the number of iterations increases the performance in an extreme way. Overall, though, results for events N₃, N₄ and N₅ may be masked as they seem to be "easy" to predict, which would hide the differences between learners. It is also interesting to note how for event N₄ there seems to be overfitting at the right part of the plot, which suggests that this is a possible risk to take into account when training this kind of models.

Figure 5.2.10: Kappa for each tested event in terms of the number of boosting iterations on the Boosted Logistic Regression models.

5.3 PERFORMANCE RESULTS

While on the previous sections I've studied how each model's parameters behaved in terms of their performance, my final objective is to select a model in terms of how well it predicts our events and how easy it is to tune. To ease this task, Table 5.3.1 contains the best performance of each model for each event.

Table 5.3.1: Best kappa values for each event and model.

Event	RF	C-CART	SVM	EN-LR	B-LR
N1	0.433	0.111	0.589	0.044	0.517
N2	0.544	0.423	0.514	0.142	0.195
N3	0.924	0.899	0.909	0.912	0.920
N4	0.872	0.831	0.873	0.651	0.850
N5	0.984	0.939	0.967	0.926	0.975
L1	0.199	0.003	0.234	0.011	0.063
L2	0.288	0.096	0.365	0.017	0.063
L3	0.360	0.052	0.410	0.077	0.006
L4	0.304	0.004	0.317	0.008	0.023
L5	0.550	0.296	0.593	0.021	0.416

The model's results are quite interesting: overall, events N1, N2, N3 and all of the L events seemed very hard to predict. In fact, no event reached 0.6 on any event of this subset. On the other ones, N4 and N5, most events obtained good results, except from EN-LR, which was left behind the others. Between the other four, cost sensitive CART also got worse performance than the other ones on most events. SVM was the most consistent algorithm, followed closely behind by Random Forests. Boosted Logistic Regression also achieved a great performance on several events, though it sank on others in which SVM or RF kept a good one. Based on these results, both Support Vector Machines and Random Forests are at the top of the algorithm ladder.

5.4 COMPUTATION TIME

One last dimension to explore is the computational complexity of each algorithm, how much it took for each to complete the training and evaluation. These results can be found on Table 5.4.1. As we already knew, Random Forests is the algorithm that takes the most to train. Surprisingly, SVM and EN-LR show the greatest variability between them all. Both CART and Logistic Regression are the most simplest ones, again, as expected.

Table 5.4.1: Computation time (measured in seconds) for each model and event.

Event	RF	C-CART	SVM	EN-LR	B-LR
N1	23238	714	2317	4335	125
N2	40266	942	3529	9492	120
N3	59747	1243	4794	3493	122
N4	62555	1046	6883	6161	108
N5	11073	873	3607	3800	115
L1	22220	2837	6566	31786	298
L2	35305	3415	8699	33115	286
L3	85479	3589	54534	17223	258
L4	99422	3732	25774	11960	297
L5	26365	3221	8246	27239	263

5.5 CONTRIBUTION SUMMARY

On this chapter I've tested five different Machine Learning algorithms with ten different events extracted from two different real datasets. I've studied their parameter dependence, their prediction performance (measured with Cohen's Kappa) and their computation time. Overall, both Random Forests and Support Vector Machines show great performance. They differ on everything else, though: Support Vector Machines are much more dependent on the tuning of its parameters to achieve a great performance. Random Forests, though, while much more costly in

terms of training a single model, shows stable performance with minimal tuning. Given this, I'd recommend using either, but taking into account the need for tuning that SVM's present. From now on, and for the sake of simplicity, all of my prediction models will be trained using Random Forests with a large enough number of trees and the Gini split rule.

6

Contribution 2: Optimizing Observation Windows

The first contribution allowed me to conclude that, for my scenarios, Random Forests was the optimal algorithm to use when training a large number of models. This contribution addresses the first open problem I found on the Failure Prediction discipline: no way to define, without previous expert human knowledge, a suitable Observation Window for a prediction model, this is, which time period to consider in the past to predict the appearance of future events.

6.1 PROBLEM DEFINITION

Based on how it is treated on previous researches, the window length problem for Failure Prediction can be formalized as:

$$\max_w P(w)$$

where $P(w)$ is a performance function that indicates how well a model is predicting failures and w is the observation window that is taken into account by the model. This model is overly simplistic and can be improved in two fronts: the first one is costs. A crucial issue in terms of model scalability (in situations where, for example, hundreds of events are to be predicted), limiting the amount of data that is utilized reduces computation and storage costs. While it may be possible to argue that Big Data solutions render this limitations invalid, storing and using too much data is not without its hindrances: there is the risk of introducing random noise into the model that may lead it to find spurious relations, which would harm its generalization capabilities when deployed on a production environment. The second area in which the original formulation can be improved is in terms of the actual window, w . I propose to change w into $W = l, w_1, d_1, \dots, d_m, w_n$ where l is the lead time, the time between the moment when the prediction is done and the forecasted moment of failure (in most papers, this is also taken as a fixed value), w_i are different observation windows and d_i are time delays between windows: time periods that the model does not use on its calculations that allow windows to be separated in time. I introduce this concept in order to allow the model to capture different relevant intervals without including any unuseful period between them: this alleviates storage costs and avoids including non important data or noise to the model. As a way of illustrating this concept, Fig. 2 shows the classical observation window versus an example of an observation window scheme that could be generated using my proposal. Adding these two improvements together, the updated formulation of the problem would be:

$$\begin{cases} \max_W P(W) \\ \min_{\{w_1, \dots, w_n\}} \sum_{i=1}^{i=n} w_i \end{cases}$$

While that is the actual mathematical expression of the problem, I will add a way of deciding the importance of both functions on the model, to allow for further

customization of my proposal to let it adapt to different scenarios.

6.2 PROPOSAL DEFINITION

6.2.1 GENETIC ALGORITHMS

Genetic Algorithms (GAs) are a subtype of Evolutionary Algorithms (EAs), which are, in turn, optimization algorithms based on the construction of generic populations and the application of biology-inspired mechanisms: reproduction, mutation, selection, etc. GAs are the most popular type of EAs. They were invented and developed by John Holland in the decades of the 1960s and 1970s at the University of Michigan. On their basic implementation, a GA is composed of a population of individuals, being each individual a random collection of bits that represent a solution for a specific problem. Then, a fitness value, $f(x)$, is calculated for each individual. $f(x)$ represents how well the evaluated solution fits the problem at hand and mimics the selection pressure the environment applies on real organisms. Afterwards, a new population is created applying the following steps: the best individuals are selected (discarding the individuals that performed badly) and new offspring is created based on the selection. After the reproduction is complete, randomly mutate the new individuals. Then, the new population is reevaluated and a newer one is created using the same steps I described. This process is repeated until an end condition is satisfied and the best solution in the current population is returned. These steps are summed up in Algorithm 1.

Most steps are explicitly left unclear, as there have been lots of ways to perform each of them. In fact, there are options even before starting to run the algorithm: the encoding of the solution is also a decision to take. As I previously said, the original encoding of the individuals was bit-based, each value being binary. But there is nothing that stops a researcher from using any kind of data representation as the input of the algorithm. The only limiting factor is that the fitness function can evaluate the proposed solution and that there are valid mechanisms for selection, crossover and mutation. Thus, it is very common to find populations of real val-

Generate n random individuals (the solutions population).

Evaluate each individual's fitness.

Do until stop condition is met:

 New population creation:

Do n times:

 (Selection Phase) Select two parent individuals.

 (Crossover Phase) Create an offspring from the selected parents.

 (Mutation Phase) Randomly mutate the offspring.

 Evaluate the fitness of each new individual.

 Check if stop condition is met.

Algorithm 1: Generic Genetic Algorithm steps.

ues, for example. Most of the variability in solutions, though, appears on steps 3.1, 3.2 and 3.3, as they are most complex ones. For instance, for selection, the basic way to decide which two individuals to cross is to randomly select them, weighting the probability of selecting each one by their fitness. This method is commonly called "roulette wheel" sampling. While it sounds coherent a priori, it presents an important caveat: as populations tend to be small in size, random selections can sometimes make bad individuals produce offspring. The first proposed solution to this problem was stochastic universal sampling, which was an improved version of the roulette wheel. Nevertheless, it still presented a problem: premature convergence. As, on early generations, the fitness variance is high, there will be a subset of individuals that present a relatively high fitness compared to their peers. If the selection is proportionate to the fitness, only these first individuals will be the ones to reproduce. This limits the search space to a narrow section. To address this, researchers have tested "scaling" methods, being Sigma Scaling the paramount example of them. Another way of fighting premature convergence is by varying the selection pressure between iterations of the algorithm. For example, early on it is convenient to let a wide array of solutions reproduce but after a number of rounds, a high pressure should be exacted to guide the solutions more strongly. A prime example of this strategy is Boltzmann Selection. Another proposed scheme to combat early convergence is Rank Selection: it consists of ranking each individual according to its fitness, which obscures the differences in fitness, and then perform-

ing a selection process on the ranked samples. Its process requires two passes on the dataset, though, which makes it slightly inefficient. An improved version of it in terms of selection pressure but more efficient is Tournament Selection. On it, a parent is selected by choosing two random individuals and a random number between 0 and 1. If the chosen number is less than a defined value, such as 0.75 or 0.8, the fitter individual is chosen. Otherwise, the other individual is selected. Lastly, there has also been a tendency to use non-generational schemes, this is, selection schemes in which the new population is not composed entirely of new individuals. Two examples of this are Steady-State Selection, in which only a small fraction of the individuals are substituted each generation and Elitism, a method consisting in forcing the best individuals to be kept, overlapping the process of selection, crossover and mutation.

On the previous paragraph we studied several proposals for performing Selection on a Genetic Algorithm. The next step is Crossover, which is probably the main feature of GAs. The most basic form of it is Single-point crossover: a single point is chosen randomly and the parts of the two parents are interchanged at that point in order to form two different, new individuals. Its natural evolution is two-point crossover, which just selects two points and random and interchanges the variables (bits) between them. Another improvement over it is just exchanging each bit according to a certain probability. This method is called Parametric Uniform Crossover. There is no Crossover method that always performs better than the other ones and finding the factors that influence the choosing of one or another is still an open problem. The last source of variability on GAs comes at the Mutation step. If the individuals are composed of bits, mutating them normally means just changing them from 0 to 1 and vice versa based on a certain probability. If the contained information is encoded as real values or characters, then the process is much more complex and, in fact, should be custom tailored to the problem at hand in order to optimize its efficiency.

Regardless of which specific schemes are used, there is always a decision to take: the specific values for the GA parameters. These are, at a minimum, the population size, the Crossover rate and the Mutation rate. While there have been efforts on

trying to establish general guidelines, the current landscape of GAs, where they are used for a wide array of problems, environments and disciplines makes any kind of effort trying to be specific to be probably useless for most situations. Nevertheless, there are some general common practices, such as keeping the population size on the range of 20 to 30 individuals and, if single point crossover is used, to have a rate between 0.75 and 0.95. There has also been a trend of finding the optimal parameters of a GA by using another GA on top of it (a meta-GA) and, also, to adapt the different parameters of the algorithm between generations.

To sum up, GAs have been shown to be powerful tools to use when there is a large search space (different combinations of possible solutions of a problem), which limits the possibility of a brute force search; and when the fitness function is complex, noisy, or unknown, which impedes using simpler algorithms, such as hill-climbing.

6.2.2 MY CUSTOMIZED GENETIC ALGORITHM

Though the basic philosophy and purpose of my proposed Genetic Algorithm is the standard one, to find optimal values of sets of variables, there are several parts of it I custom tailored to the prediction problem. Fig. 6.2.1 shows a general outline of the steps of my algorithm and which steps have been tailored to my problem. First, I will define the variables to optimize (initial population generation on Fig. 6.2.1): six observation windows, w_i , five time delays, d_j , where $i = 1, \dots, 6$ and $j = 1, \dots, 5$ and a lead time (I employed six windows as a maximum bound for the allowed window scheme complexity; this number can be increased at the cost of an increased computation time).

For its initial population, their values are generated as follows: for the lead time, I use a Poisson distribution of $\lambda = 3$ minutes with a minimum of 1 minute, a design choice that should be taken based on the amount of time needed to prepare for a failure on a specific system. The usage of this distribution is based on the following reasoning: while I assume that most failures will be close in time (which is what a low λ value generates), I want to allow the algorithm the possibility of exploring

Figure 6.2.1: Steps of my custom Genetic algorithm. Grayed out boxes indicate customized steps.

other, unexpected, solutions with higher lead times. Any other distribution that mostly generates low values with sporadic high ones would be a proper candidate for this too. Overall, the selection of the distribution should not be something that makes or breaks the general algorithm, but it could cause it to converge in more time. The second value to initialize is each window's length: for each of them, their initial values are $M_{15} + \eta$, where M_{15} is the median value of time difference between 15 events on the dataset and η is a value extracted from a normal distribution with 0 mean and $M_{15}/5$ standard deviation (my tests showed 15 events and the specified deviation rates to be large enough for the algorithm to work properly. I did not include that search for the sake of brevity. If needed, these values could be found by performing a grid search, for example). The idea behind this value is to supply the algorithm with large initial values, as, as will be shown, it will tend to shrink the windows. Thus, I expect it to start on a sufficiently large value and diminish the windows as much as possible while optimizing the model's perfor-

mance. While it may seem that this task is a matter of just shrinking the window, which could be a task easily done by simpler algorithms, such as binary search or hill climbing, it is not as simple as that. First, the problem is based on optimizing multiple variables at the same time, which renders binary search unusable. Furthermore, I assume the objective function will contain local minima, making the use of hill climbing algorithms ineffective. While this is an assumption, I err on the side of precaution, as a Genetic Algorithm will work whether it is true or not, and a hill climbing algorithm would not work if local minima exist. Growing a small window would also work, but it would need more time to converge and additional fitness and mutation criteria to avoid the windows to grow uncontrolled. Overall, it is more aligned with my objectives to shrink large windows.

The initial population created by the process described on the previous paragraph is, then, supplied to the algorithm, which, alongside the original event sequence, generates the window observations that constitute the input of the Machine Learning model (windows setup generation on Fig. 6.2.1).

My Genetic Algorithm follows the standard phases: population evaluation, individual selection, cross-over and mutation. There are several specific decisions or design criteria about them I would like to discuss. The population evaluation phase is divided in two substeps:

- Evaluation: as each individual is a set of observation window lengths and time delays, for each individual I first process the sequential data according to them, creating the aforementioned windows, and then train and evaluate a Machine Learning model using the processed data. In that way, each individual shapes how the data are preprocessed before feeding it to a (identical for everyone) training model. As I concluded on Chapter 5, I used Random Forests as the Machine Learning algorithm for the prediction models. I evaluated them using the Area Under the Roc Curve (AUC) [102] under a 3-folds cross validation scheme [103]. While a 3-folds cross validation scheme is lower than the usually recommended 5 or 10-folds, I had to settle on a tradeoff between accuracy and computation time. If accuracy is

an extremely important criterion, the number of folds could be increased. Furthermore, as each configuration is reevaluated on each generation, the impact of this accuracy loss is decreased.

- Fitness calculation: the AUC score obtained on the previous section would be a suitable way of calculating each individual's fitness. But the prediction scenario presents an issue that should be considered: collecting and processing data has an associated cost. When dealing with large scale systems, storing data recklessly can quickly turn into an unmanageable situation. Thus, I defined a fitness metric that penalizes large windows using a Window Shrinkage Factor (WSF). It is defined as:

$$F = \alpha * AUC + (1 - \alpha) * WSF$$

where

$$WSF = \frac{I_w - G_m}{G_M - G_m + 1}$$

being I_w an individual's window length and G_M and G_m the generation's maximum and minimum window lengths, respectively. This metric is reminiscent of the regularization methods for training machine learning models, whereas instead of shrinking a model's variables, my Window Shrinkage Factor (WSF) makes the windows grow smaller. Thus, depending on the costs of collecting data, I can tune how much performance is sacrificed in trade of smaller windows. I perform a study of this parameter's values on the next section.

For the selection and crossover steps I employed standard procedures: tournament selection and single-point cross-over. The mutation part, though, is also heavily customized: for each individual, there is a 5% chance that its parameters are completely randomized. If not, for each feature, there is a 10% chance of it to be mutated in the following way:

$$\hat{x} = x + (\mu - 0.3) * M_{15}/10$$

where $\hat{x}$ is the feature's new value, x is the feature's previous value, μ is a random value extracted from a normal distribution of 0 mean and 1 standard deviation. The addition of the second member of the equation, 0.3, generates a random value that will be: centered on a negative value and will have a magnitude around one tenth of the median separation between events. What this does is to, on average, shrink the windows by sensible steps, contributing to creating small windows. If the feature is 0, there is a 5% chance of it getting a random value instead. These mutation values and mean shifts were decided ad hoc by us after testing a small range of possible candidates. A standardized method of deciding their values and an analysis of their effect on the algorithm's performance is a line of future work for us.

6.3 VALIDATION

To test my approach, I must choose some other window scheme to evaluate it against. I settled on using a heuristic approach based on a grid search, as it is the only method present on literature. I used fixed values and a method based on the dataset's features to decide on the values used. I combined different lead times (1, 5 and 10 minutes) with different window combinations:

- Single windows: 5, 10, 50 and 100 minutes and M , $3M$, $5M$ and $30M$ seconds.
- Six equal-sized windows of: 5 and 10 minutes and M and $5M$ seconds.

Here, M indicates the median time separation between events on the dataset. We end up with a search body of 3 lead times and 12 window schemes, this is, 36 different experiments. I tested all methods with the two datasets I described on chapter 4, against five event types for each one. These were the selected event types:

- S1: a CPU problem on system 18 of the LANL dataset.
- S2: a CPU problem on system 19 of the LANL dataset.

- S₃: a Memory Dimm problem on system 2 of the LANL dataset.
- S₄: a Node Board problem on system 2 of the LANL dataset.
- S₅: a Memory Dimm problem on system 18 of the LANL dataset.
- N₁: critical severity. Alarms of a software issue on a virtual machine of the network on the Networking dataset.
- N₂: critical severity. Indicates a CPU threshold violation on a virtual machine of the Network dataset.
- N₃: major severity. A communication link of a virtual machine is down on the Networking dataset.
- N₄: major severity. The network router is not responding to primary management requests on the Networking dataset.
- N₅: major severity. A process or service has stopped on a virtual machine of the Networking dataset.

6.3.1 EXPERIMENT DETAILS

All experiments were run on a cluster of eight HP ProLiant SL250s Gen8 with the following specifications: two Intel Xeon e52630v2 2.6GHz (6 cores each) and 32GB of RAM. The Genetic Algorithm's parameters were: 20 individuals population, 100 generations, tournament selection and single-point crossover with a probability of 1 (I always performed crossover). We programmed it on Spark 2.2 using the Scala programming language. The Random Forests implementation I used was the one found on Spark's Machine Learning library, SparkML. I trained forests with 50 trees, a maximum tree depth of 10 and I used the gini coefficient for the impurity measures. These are the specific parameters for this Random Forests version. The posterior analyses were performed using the R statistical environment.

6.4 EXPERIMENTS RESULTS

I applied my algorithm and the grid search heuristic I described in the previous section to the five selected events from each of my datasets. I perform it once for every event to analyze, as there is no guarantee that they will show similar behavior. In fact, it would not be a surprise if they performed in a completely different way, an assumption that becomes truer as the analyzed system grows more complex. For the Genetic Algorithm part, I first obtained the window combination that yielded the best AUC of the last generation and then tested it against the whole dataset following a 10-folds cross validation scheme, repeated 5 times to reduce the potential variability of the results. Each option of the grid search approach was tested with the same cross validation process.

6.4.1 GENETIC ALGORITHM PERFORMANCE

The first thing we should study would be the Genetic Algorithm's population evolution through generations. For it to work properly, I would expect the fitness measure and, correspondingly, the AUC to increase up to a certain point and/or the window length to decrease as much as possible. For this preliminary run, I used a value of 0.9 for α .

HIGH PERFORMANCE COMPUTING DATASET

For my first dataset, I detected three different behaviors for the algorithm. For events S1 and S2 (event S1 depicted on Fig. 6.4.1, in terms of the median population behavior for total window length, selected lead time, AUC and fitness score across the generations), the algorithm starts with very high window length values, which render a high AUC. The shrinkage effect comes into play, and the window converges rapidly to a very much smaller length while only losing a trivial amount of AUC score. The algorithm's performance for events S3 and S4 follows more closely what I expected: it diminishes the window length while achieving an increase in the AUC score. Lastly, event S5 seems to follow a hybrid behavior be-

tween the two I discussed: first it starts with an extremely large window value and shrinks it very fast but, instead of then plateauing on performance and increasing the lead time, it manages to start increasing the AUC score again. One potential problem I must address is the fact that, while the AUC or the window length seem to follow a clear trend, the fitness (specially for event S_5) is much more erratic. I assume this is because the fitness function is a complex one, that combines the output of the Machine Learning model and the Window Shrinkage Factor. As its behavior does not seem to be negatively affecting the algorithm's performance, I won't address it further.

Figure 6.4.1: Genetic algorithm behavior for event S1.

As another way of checking if my proposal is working correctly, I show on Table 6.4.1 the selected window schemes for each event, this is, the individuals with the best AUC of the last generation for each event (the best fitness' individual may also be chosen). I draw several conclusions from it: as it corresponds to a dataset with a low event density (at least, compared to my networking one), the lead time

can be large without it affecting the model. This does not happen, though, for event S4. Apart from this small difference, all of the events end up with a similar window configuration: a single large window, normally around 18000 seconds, except for event S5, which employs a 28000-second one. It is remarkable how, for this dataset, the algorithm has selected a single window, not employing the multiple window scheme capabilities it has. This finding is a good sign of the algorithm's expresiveness, as it indicates that it is able to simplify the solution if the complexity of a multiple window scheme is not needed. While there are some cases in which $d_i > 0$ while $w_{i+1} = 0$, these are artifacts of the optimization process, that does not take into account the meaning of each variable. Nevertheless, the optimization process works as expected.

Table 6.4.1: Window schemes selected by the Genetic Algorithm for each event on the High Performance Computing Dataset (measured in seconds).

Event	lead time	w_1	d_1	w_2	d_2	w_3	d_3	w_4	d_4	w_5	d_5	w_6	$\sum_{i=1}^6 w_i$
S1	5947	14766	0	5275	35	167	0	63	0	0	0	0	20271
S2	11012	16298	0	370	0	0	0	0	0	0	0	0	16668
S3	1656	31	0	18450	0	0	0	0	160	0	370	0	18481
S4	60	377	0	18528	0	3	226	0	0	0	0	0	18905
S5	10854	10530	901	2538	0	14971	0	386	0	0	197	0	28425

NETWORK DATASET

Fig. 6.4.2 shows the Genetic Algorithm's behavior plots for event N2. This time, though, only the first 50 generations are shown, as the algorithm converged faster. Overall, it seems to work as expected. The window length greatly decreases, while the AUC increases. Another positive side-effect is that the lead time is also increasing, as it happened on the previous dataset. It seems to be, then, an emergent property of my algorithm. For event N4, the algorithm works along my expectations: the window length is optimized extremely quickly (in less than 20 generations) but the fitness (and, correspondingly, the AUC) is optimized until it reaches the plateau of the local performance optimum. The cause for the "jumps" on the fitness plot could be due to the algorithm trying to lower the window too much, causing

the AUC to be lowered, correcting it and then trying it to do so again. This behavior could probably be averted by setting an additional stopping criteria to the generation limit but, as it is, it is not hurting the algorithm's performance, only its computation time, which I will discuss further down.

Figure 6.4.2: Genetic algorithm behavior for event N2.

Lastly, Table 6.4.2 shows the selected window schemes for each event. At first glance, a clear observation is how different the selected schemes are, a totally different situation from the one I had on the LANL dataset. This is a good sign: even though the initial populations were generated equally for all executions of the algorithm, it has been able to adapt them to fit each specific problem, even if they are wildly different (as the differences between solutions imply). Furthermore, and in contrast to the previous dataset, this time the algorithm has chosen a multiple window scheme for several events (N2, N3, N4 and N5). This shows how there are instances where more complexity is needed to accurately predict the target event. Another interesting fact is how, for event N1, it managed to get a high lead time,

as I observed on the behavior plots previously. One point for improvement would be, though, to include a preference for higher lead times on the fitness function. As it is not explicitly included on it, my current algorithm would not select the model with a larger lead time between identical models.

Table 6.4.2: Window schemes selected by the Genetic Algorithm for each event on the Networking Dataset (measured in seconds).

Event	lead time	w_1	d_1	w_2	d_2	w_3	d_3	w_4	d_4	w_5	d_5	w_6	$\sum_{i=1}^6 w_i$
N1	1289	572	0	796	0	705	0	10	65	9	128	0	2092
N2	341	2459	55	1371	159	1415	145	1068	0	1591	269	1728	9632
N3	60	1476	75	2001	68	1321	0	1600	0	702	257	836	7936
N4	60	412	103	330	44	11	0	0	0	0	28	0	753
N5	88	1753	94	1773	235	1669	159	963	32	1276	0	1446	8880

Overall, the algorithm seems to be working as expected and, furthermore, it has shown its adaptation capabilities, as it created large single windows for the LANL dataset and varied, small, multiple windows for the Networking dataset. Another good sign is how, in hindsight, a maximum of six windows has shown to be enough to successfully model all events, as their optimal solutions are below that quantity.

6.4.2 GENETIC ALGORITHM STABILITY

While we've checked that the algorithm performs as expected on several events, we've only run it once for each of them. It is crucial to ensure that the algorithm is stable between runs, both in terms of performance and in terms of the chosen window scheme. To do so, I ran the algorithm ten times for the N4 event. There is one more thing to consider, though: the effect of α , the Window Shrinkage Factor parameter. As it alters the fitness function, it surely affects the algorithm's stability. So I expanded my experiment and ran the algorithm 10 times for $\alpha = \{0.75, 0.8, 0.85, 0.9, 0.95\}$. Fig. 6.4.3 shows the density of the median results for the individuals of the last generation of each run. For other values of α (0.75, 0.8 and 0.95) I observe too much dispersion in terms of either the AUC or the window lengths. Both 0.85 and 0.9 seem to be stable enough to be used. I also ran

Kolmogorov-Smirnov tests on the obtained performance distributions and confirmed that they can be assumed to follow a normal distribution. So, I consider the algorithm stable in terms of performance for these values.

Figure 6.4.3: Genetic algorithm stability for 10 runs of different values of α in terms of median AUC, median fitness and median window length of the individuals of the last generation.

There is one last area I must study to confirm that the algorithm is stable, though: the specific window configurations it chooses. To test this, I took the 10 runs from the $\alpha = 0.9$ test for the N_4 event and did the same for the S_1 event. Then, I took the individuals of the last generation of each run and plotted the obtained schemes. These are shown on Fig. 6.4.4. N_4 's results are clearer: the algorithm always focuses on the same period (while there are some mutation remnants that focus on different periods, they are a negligible quantity). S_1 's results are trickier: while it may seem that there are two different behaviors, I must consider that the only thing relevant to the algorithm's performance is whether it captures the events

that influence the target event's appearance. In this sense, I can observe that both behaviors overlap on a certain time gap. This leads us to think that the important events happen on this time interval. In fact, studying these window configurations would be something very interesting that could lead us to reduce the observation windows even further. This could be a promising line of future work.

Figure 6.4.4: Window configurations for the individuals of the last generation of 10 different Genetic Algorithm runs for events S1 (left) and N4 (right).

Based on the two experiments we've shown, I consider the Genetic Algorithm stability has been proven. In order to compare its results, from now on, I will only

run the algorithm once.

6.4.3 AUC RESULTS

Having confirmed that the Genetic Algorithm is working as intended, I have to check how it compares against other solutions in terms of model performance. This is, how well the Machine Learning model trained with the selected window scheme performs against other similar models trained with different window schemes. As, regarding storage, what matters is the total combined observation window that will be analyzed, I will assign a total window length to each window scheme and plot the performance results along this axis.

HIGH PERFORMANCE COMPUTING DATASET

Fig. 6.4.5 shows a summary of the algorithm's performance against the other fixed schemes in function of the total window length. The results were obtained averaging the obtained performances by window size. The variances were negligible. There are, again, three different behaviors of the algorithm, regarding how it compares to the fixed values models performance. Events S1 and S4 show similar results: the algorithm is able to achieve higher performance than similar window values for the fixed approach, being lower, though, that the AUC obtained by the extremely large window options. Events S2 and S5 perform even better: even using relatively small windows compared to the fixed schemes options, the algorithm performs better than all of them, obtaining extremely high AUC scores. The only event in which the algorithm does not outperform the fixed schemes in one way or another is event S3, which appears to be a harder to predict event, for both the fixed schemes and for my algorithm. Another interesting observation that can be made from these graphs is how the multiple window scheme works worse than the single window scheme for this dataset. This is something that was hinted previously by the Genetic Algorithm's chosen schemes and is confirmed now. Additionally, in order to compare my proposal's performance against each single other scheme, I took all the cross validation raw values (the AUC obtained on each fold for each

model) and performed an ANOVA and a Tukey's range test [104], that runs one on one comparisons between the different schemes and gives an estimation of the difference between values and a p-value to indicate if the difference is significant. For this dataset, the results were good: out of the 36 schemes against I tested the proposal, it was (in average) better than 27 of them and worse than 4 of them, with no difference to five of them. This result is supported by what I just studied: extreme window values yield a higher performance, but my method finds a sensible trade-off between performance and storage costs.

NETWORKING DATASET

For the Networking dataset, the chosen schemes were closer in total window length and the obtained variances were not negligible, so I chose to plot them as boxplots. For the sake of clarity, I only chose the best five schemes for each event. Fig. 6.4.6 shows the AUC results for each of them. The first insight that can be extracted from the figure is how the optimal window and behavior based on it of each event is completely different (ranging from 750 seconds to more than 12000), being exactly the same training dataset. This observation suggests that, for certain datasets, a single, fixed value window for a whole dataset is not a suitable approach and it would yield subpar results. Regarding the Genetic Algorithm results, it works as intended. For events N₁, N₂ and N₃ its solution obtains either as good a solution as larger schemes with less window length or it manages to stay between them, shrinking the window length until performance would start getting lower. But events N₄ and N₅ show another very interesting behavior: in both cases, the Genetic Algorithm solution obtains the best median performance with a smaller window than most of the other ones. This suggests that the algorithm has, indeed, found a complex distribution of windows that skips useless time periods and keeps the ones that contain model-relevant information. In order to check this behavior, I show on Fig. 6.4.7 the Genetic Algorithm window for event N₄ and its most similar instance in window length, a 60s of lead time and 600s single win-

dow scheme. I recommend, though, to run a similar analysis to the one I show on Fig. 6.4.4, in order to find the consensus of the algorithm and avoid the dispersion caused by random mutations. I also performed an ANOVA and a Tukey's range test to check if the Genetic Algorithm's results were significantly better than other schemes. The algorithm worked much better for this dataset than for the previous one: out of 36 different schemes, for the five events, my proposal was better than 25 (with a minimum of 21 for event N1 and a maximum of 30 for event N5) schemes in average (with a mean median difference of 0.14 in favor of it). For the rest of the schemes, there was no significant difference. So, even optimizing the window size, my proposal worked better (in raw performance) than the great majority of schemes tested.

There is also one area where the heuristic approach works much worse than my proposal: computational complexity. Its complexity can be defined as:

$$GP(RO(ML) + O(Fitness) + O(Selection) + O(Crossover) + O(Mutation)) \sim GPRO(ML) \quad (6.1)$$

where G equals the number of generations, P is the chosen population number, R is the number of cross-validation rounds and $O(ML)$ is the complexity of the chosen Machine Learning algorithm. As the number of parameters and values to test increase, the computation time of a grid search increases to unfeasible amounts: for example, for my setting and the High Performance Computing Dataset, one execution of my proposal takes around 35 minutes, whereas the grid search (using 36 different schemes) took around 7 hours and 30 minutes. Additionally, there's the fact that a simple grid search doesn't allow for complex windows, which I saw appear on the Networking dataset.

Figure 6.4.7: Windows selected for event N4 by the Genetic Algorithm and its closest heuristic counterpart.

6.5 CHAPTER SUMMARY

As we saw on the System of the Art review, there is no current solution to decide the appropriate window size and most efforts on the field only study how the window size influences their problem at hand. To solve this issue, I have proposed a novel approach, based on the usage of multiple observation windows, separated by variable time delays, automatically optimized by a customized Genetic Algorithm. This algorithm finds the solution that maximizes model performance, while trying to shrink the window size as much as possible, to save storage costs. As systems grow in complexity and the number of events to be modeled also grow, the saved storage becomes more and more significant. I have validated my solution against ten different events, extracted from two real datasets, showing how the algorithm has consistently performed equally or outperformed a heuristic grid search, finding complex window schemes (and single windows when they were needed, which shows the algorithm is not biased in favor of multiple windows) and greatly adapting its solution to each event. While there are several caveats and areas which could conform future work, like the dependency of the variables or devising a better stopping criterion, on its current form my Genetic Algorithm solution provides a complete solution to the windowing problem.

Figure 6.4.5: AUC vs window length results for the Genetic Algorithm versus the heuristic approach for the High Performance Computing dataset.

Figure 6.4.6: AUC vs window length results for the Genetic Algorithm versus the heuristic approach for the Networking dataset.

7

Contribution 3: A Method for Machine Learning Based Root Cause Analysis

This third contribution tackles the problem of extracting information about correlations between failures and events in the modeled system: can we know what are the probable causes of failures? As we saw on Chapter 2, the standard method to do so, Bayesian Networks, requires previous knowledge of the system, so it does not fulfill one of the main requirements of the ideal Failure Management system. This chapter maps to reference [14].

7.1 PROPOSAL DEFINITION

I propose a method that allows for the construction of an offline diagnosis helping system based on a Machine Learning algorithm (in my example, Random Forests,

though almost any algorithm can be used) and the analysis of its results by summarization and graph theory, that does not require deep knowledge about the modeled network and allows for the inclusion of several temporal scale ranges through windowing on the feature engineering phase.

Fig. 7.1.1 sketches my proposal's workflow: the upper boxes represent the intermediate and final information obtained by the application of the operations in the lower part. It is organized as a pipeline of information, starting from system observations and finishing with causes of errors, getting from the former to the latter through the training of Machine Learning models. While their usual purpose is predicting new events, I intend to use them for information extraction. I do so by exploiting a concept named "variable importance", a measure of how much a certain feature is contributing to accurately predicting the objective variable on a model. Some specific algorithms have their own importance measure (such as variable coefficients on a regression) but some general proposals have also been created [105] We, thus, train a set of models for every event on the system, extract every model variable importance and create the cornerstone of my proposal: the influence matrix, that represents in a compact way the obtained information. If we have n events to predict and m events to use as variables (with $n \leq m$), and $i_{i,j}$ is the influence of event j on the model for event i , the Influence Matrix, I , is defined as:

$$I = \begin{pmatrix} i_{1,1} & i_{1,2} & \cdots & i_{1,m} \\ i_{2,1} & i_{2,2} & \cdots & i_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ i_{n,1} & i_{n,2} & \cdots & i_{n,m} \end{pmatrix}$$

This measure of influence of an event over another one points to the concept of causality. Although it cannot be said that an observation of system behavior is the cause of another one, if the Machine Learning method determines a certain degree of influence, we can use it to represent how the appearance of an event contributes to the appearance of another one. So, we can consider the value in each matrix cell the weight of the vertex from the event in the column to the event in the row and

study it both as a matrix and as a directed graph. Furthermore, there is a myriad of directions and perspectives we can take to analyze the system just by extracting information from the Influence Matrix.

Figure 7.1.1: Proposed RCA methodology.

7.2 VALIDATION EXPERIMENTS

On this section, I will show several ways to analyze the Influence Matrix, starting from a system perspective, going further down until getting to single specific events, as well as extracting different summarization tables and indices.

7.2.1 EXPERIMENT DETAILS

To carry out these experiments I used my own Networking dataset: it is a large enough dataset with lots of different errors and a plethora of events of different importances. Additionally, as the dataset contains each event's severity and location, it is a perfect candidate to show how the Influence Matrix can be enriched when more variables are added to the dataset. To start the analysis, I trained 501 Random Forests models following the usual windowing process I previously studied, using a 5 minute observation window (the reason for this specific number is because not every event was preceded by other events and only events preceded by others can be analyzed in search of event correlation). I decided to use a 10-fold cross validation scheme, with the SMOTE algorithm [9] for class imbalance and the Area Under the ROC Curve [10].

7.2.2 MODELS VALIDATION

There is a previous, needed step to the creation of the Influence Matrix: the validation of the patterns extracted by the models. If the information they convey is not accurate, then the Matrix should not be trusted. This accuracy is proved, of course, by checking the AUC score, as it indicates each model's generalization capabilities. Additionally, there is another potential problem: lack of samples. If there were not enough original samples in the dataset of a certain event, for example, the model would not be able to be trained correctly. In fact, from the 501 generated models, 151 of them were not correctly generated, leaving 350 models left. Their AUC distribution can be seen on Fig. 7.2.1. To ensure the quality of the extracted information, I imposed a 0.8 threshold on the AUC score, which filtered out 18 models. This value could be tuned to the necessities of the problem at hand. The final number of models used to create the system influence matrix was 332, so the resulting matrix contained 332 rows (one for each model) and 548 columns (corresponding to the number of events that influence each model). While almost a 34% of the models were lost in the filtering process, these were events for which there was not enough information or the training process did not perform correctly. For the specific algorithm I used, Random Forests, this variable importance of each feature is calculated as the difference in prediction accuracy between training it normally and retraining the model when the variable is replaced by random values. The rationale behind this process is that if the influence of an event is not important, randomly permuting its values should not affect the predictor's performance. If it changes, we can measure how much it does and that gives us an indication about its overall importance in the prediction process.

Figure 7.2.1: AUC distribution of the trained models.

7.3 ANALYSIS OF SYSTEM-WIDE PROPERTIES

A first, visual assessment of the influence matrix can be useful in order to find general patterns. One way to visualize large matrices is to draw a heatmap of them (Fig. 7.3.1), where each row represents an event model and it contains the influence values of the others on it. Both rows and columns have the events in the same order, events in the same device appear together ordered by their severity. Vertical lines show the influence that a certain event has on all the models and horizontal lines indicate how the event they represent is influenced by the others.

Figure 7.3.1: Influence Matrix depicted as a heatmap.

There are several observations to extract from the heatmap:

1. There is a large part of the matrix (almost 75%) with non significant values (yellow); I interpret that only a fourth of events convey information useful for RCA. Empty rows signal events for which no cause can be discovered. Empty columns are for events that do not add information. This suggests that, in average, models are influenced by a small number of events and vice versa.
2. There seems to appear a principal diagonal, which suggests the appearance of most events is influenced by the appearance of the same event at a time lower than the observation window. Perhaps these represent alarms on certain conditions of the network, that once they are raised, generate events

periodically.

3. The switches seem to have a strong influence on themselves and on several virtual machines. The virtual machines work in groups. The router has around 5 events which strongly influence everything else, something that remarks its importance on the network.

Direct manipulation of the influence matrix allows reasoning on events, aggregated by severity, rendering a summarized table (Table 7.3.1) whose rows and columns are severity levels and each cell contains the median proportion of events of the row's severity affected by events of the column's severity.

Table 7.3.1: Aggregated matrix for the severity of events: each cell specifies the median proportion of events of the row's severity affected by events of the column's severity.

	Blank	Minor	Major	Critical
Blank	0.13	0.38	0.11	0.52
Minor	0.14	0.11	0.14	0.29
Major	0.11	0.29	0.14	0.33
Critical	0.10	0.30	0.10	0.80

I draw two important conclusions from Table 7.3.1: Critical events affect, overall, the largest proportion of events, which is consistent with the fact that this kind of events is supposed to cause a large disruption in the system. And Critical events only seem to be affected, mostly, by other Critical events. This is a potential issue in terms of network resiliency which would point to a need for an event policy recollection improvement.

A different, complementary analysis on the influence matrix is to consider it as a graph where nodes are types of events (type-device, in fact), and edges are influence values. Thus, from an influence matrix I , composed of m rows and n columns, each cell containing $i_{j,k}$, the influence value of event j on the model for event k , we can obtain a graph G composed of a set of vertices V and edges E , where V is the

set of events that appear on I and

$$E_{j,k} = \left\{ \begin{array}{ll} i_{j,k}, & \text{if } \exists i_{h,j} \\ \emptyset, & \text{if } \nexists i_{h,j} \end{array} \right\}$$

This kind of analysis permits obtaining an “Influence Topology” of the network, even if no topology information is present on the dataset. This can either confirm that the actual topology dictates most of the devices’ behavior (such as: separated devices don’t influence each other) or it can lead to discovering some cascading problem events or situations. While such a graph can be directly plotted, for most systems this would lead to an entangled mess, with no clear knowledge or relation present. So, I filtered it by edge magnitude (influence magnitude between events), solving the visualization problem and displaying only the strongest relations in the system. The selected edge magnitude threshold can be varied depending on its size, the effect magnitude distribution and the desired strength of the relationships shown. I propose to use percentile analysis to set this value, as it allows the user to abstract from absolute values. Using this, two graphs were constructed: first, I used the top 1% influences on the influence matrix. Analyzing it using Gephi and the algorithms it implements, I extracted some graph metrics from it that give us some insights on the system: it is composed by 381 nodes and 1614 edges, forming two disjoint graphs. Its average node degree, the average number of edges (input or output) per node, is 4.2, meaning that, in average, each event is related with around 4 different events.

Figure 7.3.2: Influence graph of the top 1 percent influences, including communities.

Employing the visualization capabilities of Cytoscape, Fig. 7.3.2 was generated, which shows the graph, along with the different communities I found applying the algorithm proposed in [11]. Thus, for example, there is a group of events related with alerts and threshold violations; another one related with devices connection problems; maintenance operations; execution of OSPF in the router; and the stopping of Java management agents (these were obtained inspecting the different communities' members). This visualization shows the two different disjoint graphs (the second one being the small community on the lower left part) and how, generally, critical events are associated with large clusters of blank and major events, suggesting that they cause a large disruption on the system. There also seems to be an overall separation between hardware devices, where switches are mainly influenced by other switches and virtual machines are influenced by the router or another virtual machine. One observation I must remark is that, of course, the events I am studying are the events I predicted on the Model Comparison chapter. Thus, we can see how this proposal complements that kind of

analysis: whereas on that chapter I strove to predict events as accurately as possible, using the Influence Matrix I can try and see which events preclude a critical failure by more than one observation window. In this sense, this proposal is built on that one and takes it further, from a single model perspective to a system one.

Even though the rich analysis we studied on the previous paragraph can give a broad overview of the system, the information contained in the data can be leveraged to view it from a different angle. Specifically, summarizing all the device information while excluding the events allows us to study the general device behavior on the network. As this resulting matrix is much smaller, we can visualize its top 10% influences as a graph, Fig. 7.3.3. It shows much clearly the hardware division I spoke about previously: switches interact mainly with each other and the router is the one that influences events on virtual machines. As a novelty, this graph unveils how virtual machines are normally paired with one another, probably indicating that they are being executed on the same physical device. Also, the router is only influenced by itself and, in turn, influences both switches and virtual machines. This figure also offers a clearer view of something that the previous one and the heatmap hinted: the switches form a highly interconnected mesh of devices, suggesting that a failure on one of them may render the whole network unusable, whereas failures on virtual machines don't seem to propagate to each other except to their pair.

Figure 7.3.3: Summarized devices graph.

7.4 IDENTIFICATION OF CAUSES OF ERRORS

Once a system-wide characterization has been made, if necessary, specific events can be studied. As an example, focusing on certain critical events, there are several interesting observations I can take looking at the graph on Fig. 7.3.2 and their event descriptions:

1. From the isolated critical event on the left (arrow number 1) I find that losing contact with the virtual machine manager precludes a virtual machine stopping to respond to requests.
2. From the group of three critical events on the lower part of the graph (arrow number 2) I observe that a critical event (a software alarm) on any of the three virtual machines causes a plethora of blank, minor and major events.
3. On the top left part of it, a large cluster of critical events (arrow number 3) on switches alarming of loss of contact to them is surrounded by a myriad of management events and to a loss of contact on a specific virtual machine.

Overall, these observations show how, usually, critical events are highly disruptive situations with a lot of blank events associated to them. But, on a system of this scale, just inspecting the graph can be cumbersome in order to find specific event causes or influences. To do so, we can perform a search on the matrix for all the associated nodes of a specific event. This gives us a smaller, clearer graph that can be leveraged to analyze an event in depth. An example of this highly focused analysis is shown on Fig. 7.4.1, that contains the strongest influences on a critical event in a specific virtual machine alarming of a CPU threshold violation. The dashed line shows a chain of events of normal behavior: when a software is started or some process changes on the virtual machine, there is a sudden spike of CPU signaled by, first, a major event and, then, a critical one. On the contrary, the arrowed line shows a strange situation: a chain of a blank and a major event on the router indicating that it is not responding to SNMP messages is influencing the critical threshold violation and not the major one. This fact tells us that most of the time that the major violation happened, the router event was not an influence of it. Thus, there is an anomalous situation, different from the expected behavior of the virtual machine, that causes its CPU to spike when the router suffers this specific event. This would need further research in collaboration with the network managers.

Figure 7.4.1: CPU threshold violation critical event chain influences.

7.5 MISCELLANEOUS ANALYSIS

Lastly, on this section I'll expose how the Influence Matrix codifications lends itself to a multitude of approaches, all of them different in terms of scope and objectives.

7.5.1 DANGER INDEX

First, I would like to propose another highly summarized index to analyze the system, the Danger Index. It is a normalized value that indicates how "harmful" an event is for the system, calculated as the percentage of Critical and Major events that are strongly influenced by each specific event. So, it can be calculated as:

1. Being e_{c_k} a model for a Critical event k , with $k \in \{1, \dots, n\}$ and n the number of Critical event trained models.
2. Being e_{m_j} a model for a Major event j , with $j \in \{1, \dots, m\}$ and m the number of Major event trained models.
3. Being $i_{h,l}$ the influence of event h on the model for event l .

4. Being α a threshold influence value, defined by the user.

5. Being $\delta_{h,j} = \begin{cases} 1, & \text{if } i_{h,j} \geq \alpha \\ 0, & \text{if } i_{h,j} < \alpha \end{cases}$

6. The Danger Index for a certain event h , DI_h , is defined as:

$$DI_h = \frac{1}{n+m} \left(\sum_{k=1}^{k=n} \delta_{h,k} + \sum_{j=1}^{j=m} \delta_{h,j} \right)$$

It is a similar approach to the severity table I studied before, but this time it is specific for each event and I only take into account values above a certain threshold α , to ensure that the influence they have on events is important enough. In order to keep this analysis general, I decided to set α using quantile values. To further explore this index and its behavior, I calculated it for different values of $\alpha \in \{0.9, 0.95, 0.99\}$. The different Danger Index distributions obtained are shown on Fig. 7.5.1. As it is expected, the Danger Index values decrease as I restrict the influence values to higher quantiles. For the first distribution, the top 10% influences, more than a 40% of the values are 0, more than 200 events that do not strongly influence any Major or Critical event. On the other hand, there are 8 events with a Danger Index higher than 0.5. This is, there are 8 events that influence more than half of the 55 Major and Critical events on the system, at least, as a top 10% event. And, in fact, all of them are Blank except for a single Minor event. This finding suggests the presence of several innocuous-seeming events (as shown by their assigned severity) that are, in fact, precursors of potentially system-threatening errors. Of course, this Index has the same drawback than studying models on their own has: it only takes into account a single observation window in the past, whereas the correlation chains I previously studied (Fig. 7.4.1) allow, in a simple way, to study a single event in much more detail. These two analyses, then, complement each other: the Danger Index can help to quickly pinpoint potential harming events, while the correlation chain analysis creates a longer chain that would allow network operators to adequately prepare for a future error with

enough time ahead. As a side note, the event that has the higher value on the top 1% distribution is the event that signals a change in the configuration of the router. This fact suggests that, probably, a lot of errors were produced based on manual changes by the network operators (this is supported by the fact that I know the dataset was extracted from a testbed environment). I remark this fact to illustrate how this simple analysis can shed light into, not only which events are frequent and dangerous, but also their possible cause, as well as potential pieces of advice for the network managers (e.g. create a special category for manual changes so they can be more easily detected and filtered).

Figure 7.5.1: Danger Index distribution for the events on the system for $\alpha \in \{0.9, 0.95, 0.99\}$.

7.5.2 MEAN INFLUENCE INDEX AND NUMBER OF AFFECTED EVENTS

Another easy to calculate index that can shed some light on how events behave on a system level is the Mean Influence Index, MII calculated as

$$MII_j = \frac{1}{m} \sum_{l=1} l = mi_{j,l}$$

which indicates, on average, how much influence the event has had on each model it has affected. If we combine this value with the Number of Affected Events by a certain event j , calculated as

$$NAE_j = \sum_{l=1} l = m\delta_{j,l}$$

$$\text{where } \delta_{j,l} = \begin{cases} 1, & \text{if } i_{j,l} > 0 \\ 0, & \text{if } i_{j,l} = 0 \vee \nexists i_{j,l} \end{cases}$$

we get a two-axis representation (Fig. 7.5.2) that allows the characterization of events in different, determined behaviors. I draw two interesting observations from this plot: there are several events that affect a small number of other events but do so very strongly (with an almost equal distribution of Blank and Major events) and there are two Blank events that affect a great number of events with a fairly high influence level. The event with the highest value is a Major event alerting that the power supply of a switch is on critical condition. It sounds very sensible to expect it to be a clear indicator of a larger problem if left unattended. On the other hand, the first of the pair of Blank events that have a high influence index, the one that affects 26 events, is the router reporting a change in network topology. Again, this suggests a manual change in the network performed by human intervention. These are, again, examples of simple analysis that the Influence Matrix form allows that are quick, easy to perform and understandable with no special knowledge needed.

Figure 7.5.2: Event distribution based on their Mean Influence Index and Number of Affected Events for Major and Critical events.

7.6 CONTRIBUTION SUMMARY

Root Cause Analysis is a key part of networks. Current solutions based on Bayesian networks suffer some limitations, such as a high dependency on previous knowledge or some limitations when representing temporal relations. As an alternative, I proposed a method that allows users to decide which kind of relationships and modeling they want to use for their analyses, while offering a highly compact matrix that contains the network's behavior. I have also shown several applications and summarization tools that can be used or applied to it, including several novel approaches, such as expressing the influence matrix as a directed graph or proposing an Event Danger Index, which allows network managers to quickly assess how the network behaves. my methodology forms a toolset of analysis that can extract useful data from networks even when little to none information is available apart from the dataset. As an example, I applied it to a real dataset obtained from a corporate network, to discover several nontrivial facts about it on a sys-

tem overview level, such as the distinct influence separation based on hardware, where the switches are mainly influenced only by themselves whereas the virtual machines are paired. I also analyzed the network on a specific level, such as observing how critical events cause major disruptions in the system, and studying the influences of one single critical event, discovering an anomalous influence caused by the router. I have thus shown how my proposal not only allows for probable cause extraction but also serves as a guide for further diagnosis researches on the network. It can be applied to any network dataset that can be expressed as combinations of events and timestamps and it can work with large networks, due to the summarizations included in the workflow. Furthermore, in large scale networks, matrix factorization techniques such as Singular Values Decomposition could be applied to the influence matrix in order to subdivide the problem into smaller, independent problems, allowing my proposal to work in a modular, efficient way.

8

Contribution 4: Online Failure Prediction Implementation on a Big Data Platform

This chapter details the last contribution of my thesis. While the previous ones were purely scientific ones, this one is more focused on an industrial issue and has two objectives:

1. To validate in a realistic environment the ideas behind the Failure Prediction models we've discussed on previous chapters.
2. To propose an implementation solution for an open issue: how to design, develop and test an Online Failure Predictor using current Big Data solutions under realistic conditions, as well as presenting the requirements and constraints that can be encountered.

The bulk of this chapter will thus be the details of the implementation my team and I designed. The rest of the chapter is divided based on the different parts of the system: an overview of it, detailing the specific software and hardware I used, the Training Subsystem, including the specific evaluations that I ran to test the Machine Learning model I used, and the Prediction Subsystem, which contains a discussion about the problems that arise when trying to run hundreds of online models in parallel. This chapter's contents were published on reference [15].

8.1 DESIGN PROPOSAL

The first thing I'd like to do is to specify the objective of the architecture I will be detailing later, as it will inform the options and decisions I took when designing the whole system. My objective was to create a piece of software that would be able to be plugged to an already running network using some kind of management system that created network events, store these events, create Machine Learning models that predicted them and, then, executed these models in an online fashion in order to help network operators managing the network. As I intended for the system to be able to be used in industrial, realistic conditions, there were several imposed constraints:

1. Scalability: the architecture must be able to store, process and execute large amounts of data and, potentially, hundreds or thousands of Failure Prediction models at the same time.
2. Reliability: the Machine Learning models that the system trains must be reliable enough to be deployed on a real system. False predictions can be extremely dangerous depending on the system they apply to.
3. Minimal latency on the online part of the system. If it has to issue failure predictions each period, there can be no lag.
4. Based on industry standards, in order to maximize the ease of plugging the system to a network.

5. Open Source: while this is not a hard requirement, showing that an industrial-level solution can be designed completely based on open source software has value by itself.

Taking into account all of these issues, Fig. 8.1.1 contains the architectural design of my proposal. The Training Subsystem performs offline computations from past event data collected from the management system, while the Prediction Subsystem performs Online Failure Prediction based on the events captured by the management system and the trained classification models. The input to the system is the stream of events obtained by the management system, rather than network metrics or resource usage. This way, the prediction system is isolated from the managed network, as only a direct communication socket between the predictor and the management system is required. On each subsection of the chapter we'll detail each subsystem but, for now, let's detail the software I chose to use as the base processing platform: Apache Spark 1.6.0. It provides a dataflow programming model with in-memory computations, automatically distributing Resilient Distributed Datasets [106], and processing them in parallel across the computation cluster. Spark also supports a micro batch stream processing model through Spark Streaming [107], where events from a stream are automatically aggregated in computation windows and processed generating an output stream. While the microbatch model introduces a minimum delay (about 1 second), this time resolution is appropriate to perform event ingestion for network and system management purposes, for prediction in particular [108]. Apart from Spark, I used HDFS (Hadoop Distributed File System, the current standard on distributed storage for large files in number and size) for the storage of the data that would be fed to the training subsystem, Kafka to communicate and create a queue system with the event manager and Spark MLlib to train and deploy the Machine Learning models. This library has been shown to be the most appropriate to implement stream-based Machine Learning systems over other similar ones such as Apache Storm or SparkML [109][110]. I must remark, though, that while Spark abstracts from many complexities of writing parallel programs, implementing complex processing flows that take large amounts of data requires significant tuning and exper-

imentation [79].

Lastly, I chose a dataflow architectural pattern to develop the prediction system, as it nicely fits to the transformation of streams of events that the system get as input. Data flow architecture defines the data ingestion and the sequence of transformations to be performed up to the final result. The event streams generated by the management system are aggregated and sliced into observation windows; the prediction subsystem predicts future events in the lookahead window based on this input.

Figure 8.1.1: Online Failure Prediction System Architecture Proposal.

8.1.1 TRAINING SUBSYSTEM

This subsystem is represented on Fig. 8.1.1 by the upper gray box. It takes as an input the historic dataset of events, performs the windowing process I studied on previous chapters and then trains one prediction model per event that the system administrators desire to predict. A summary of this process is shown on the left part of Fig. 8.1.2, where w_o stands for "Observation Window" and w_p stands for "Prediction Window". There are two observations I'd like to make about this figure. First, the system uses a sliding window system: it groups the event stream into micro batches, which are by themselves the different observation and prediction windows, and then feeds them to every model generator. And that is the second

remark: every model is fed the exact same windows, so there is no calculation overhead in this step depending on the number of trained models.

As Fig. 8.1.1 says and as I've also remarked before, I used Random Forests as my Machine Learning algorithm, including the ones I trained for this system. As I showed on my first contribution, they perform very well with minimal tuning. There was one issue this time, though: all of my previous tests were done using the R language and its Random Forests implementation. This time, I was using Spark MLLib, which had different parameters to tune, so I had to evaluate them. The next subsection details these analyses. I must also remark that training a set of RF models is a complex and computationally expensive task. That's why I took advantage of RDD transformation parallelization to speed up its completion. Additionally, the trained models can be trusted to keep similar prediction accuracy, as long as the event source does not experience significant drift, assuming the model was trained on an information rich dataset with enough samples, temporal extension, number of appearances of each event type and relationships between event types. Including mechanisms to adapt the models in real time to the normal drifts and changes of the system would be a very interesting next step that could be tackled in future investigations.

Figure 8.1.2: Training Subsystem and Prediction Subsystem execution examples.

RANDOM FOREST VALIDATION

On this section I'll explore two things: first, the effect of the Random Forests I'm using (Spark MLLib's) parameters on the performance, measured using the F_1 score. Additionally, I'll evaluate the best F_1 score obtained for each model, in order to filter out any non reliable one. So, first of all, the specific parameters the Random Forests implementation of Spark MLLib has are the following:

- Number of trees: this parameter is self-explanatory and equal to the one I tested on Chapter 5. Based on my previous tests, I settled on using 50 trees as a good-enough value.

- Feature selection strategy: the number of randomly selected variables each tree will be grown with. There are four options: use all features, a third, the squared root of the total number or the $\log_2$ of it.
- Splitting criterion: this parameter was also explored previously. It can split based on the Gini impurity or based on an entropy calculation.
- Maximum tree depth: one of the two stopping related parameters, this one limits the number of node layers each tree can have. It can range from 0 to 30. Literature suggests setting this value close to the binary log of the number of features, which renders $\log_2 575 = 9.16$. I experimented on the $\{3, \dots, 17\}$ range, as higher values did not render any significant improvement.
- Minimum information gain: the second stopping related parameter, it stops the tree growth if the split does not improve the gained information in, at least, a predefined value.

In order to find the optimal combination of parameters, I trained a model for a Critical event with each set of value combinations and evaluated it using a 4-fold cross-validation scheme. The average results are shown on Figs. 8.1.3 and 8.1.4. The first clear observation I can make is how the two impurity criteria don't seem to impact performance too differently, a result that agrees with my previous experiments. Secondly, the feature selection strategies based on taking every variable and one third of them seem to dominate over the other pair. Thus, I focused the analysis just on them in order to appreciate whether they showed any difference. They don't seem to behave much differently when it comes to performance and training time, though the "all" strategy converges much faster (both converge around a tree depth of 9) and maintains a smoother computation time profile. Based on this, I settled on it over the other strategies. In the end, the combination of parameters that I used for the rest of the experiments is shown on Table 8.1.1.

Figure 8.1.3: Average F1 score for the models based on tree depth, feature selection strategy and splitting criterion.

Figure 8.1.4: Average F1 Score and training time based on tree depth for the two best feature selection strategies.

Table 8.1.1: Optimal combination of parameters for SparkMLLib's Random Forests implementation.

Parameter	Value
Number of trees	50
Maximum tree depth	9
Feature subset strategy	All
Impurity criterion	Entropy

Lastly, I took the selected parameter combination and trained every model of my dataset with them, validating under a 10-folds cross validation scheme. Fig. 8.1.5 shows the cumulative plot for the obtained F1 scores. Out of the 575 models (corresponding to every device-event type combination present on my dataset), only 108, a 19%, yielded a good enough score (I set the threshold at 0.75 but, of course, this value should be decided alongside each specific network's managers). This kind of simple analysis gives us the power to easily rule out unreliable models and ensure that the models that are actually deployed can be trusted.

Figure 8.1.5: Average F1 Score obtained for each model. Color indicates the model severity: Blank (gray), Minor (green), Yellow (major) and Critical (red).

8.1.2 PREDICTION SUBSYSTEM

This is the second main part of the system. It takes as an input the trained models (from the Training Subsystem) and the current observation window in order to

perform predictions on the stream of management events (corresponding to the lower part of Fig. 8.1.1). It must be noted that this phase operates online, in contrast with the offline training phase. The stream of events is fed to the prediction subsystem through Kafka queues. This is specially important because it allows the system to be deployed on different environments, including ones where each specific device emits its own events. This flexibility comes from the fact that Kafka creates pub-sub channels. This permits any number of subscribers to include their own events. While this feature is not used on the specific system I worked on, nothing would need to be changed in order for it to work on such kind of environments.

The prediction flow is shown on the right side of Fig. 8.1.5. Each time an observation window is created, and for the time it lasts, Spark Streaming sets a data chunk with the events happening during that time span. The system is designed to issue a new prediction each second for each of the 108 events of interest - as a compromise between prediction granularity, machine load and capacity to produce understandable results. Each one of the 108 RF-based event prediction models runs as an independent task that predicts the appearance of one type of event. Each one of the 108 predictions must be made within the one second window, otherwise the online system becomes saturated. Each prediction window starts right after the end of its observation window. This way, each second, 108 positive or negative predictions are obtained about the appearance of each event over the next 5 minutes (prediction window), based on the previous 5 minutes (observation window); the combination of window and prediction intervals means that, for each event of interest in a given second, 300 predictions are open. Interpreting the semantics of multiple predictions for the same event at once can be confusing for human operators, increasing the need for adequate visual representations. Because of that, we designed a visualization component in order to provide system administrators with a global overview on both past state and future predictions. The aim of this visual component is to guide system administrators on corrective actions such as killing a process, stopping a node or launching a new Web server instance.

Figure 8.1.6: Visualization tool example.

Fig. 8.1.6 shows part of the visualization prototype my team and I developed to help in the interpretation of results. The x axis shows time, with the vertical line showing the current time. Different events (type and node) appear at different heights, with horizontal bars representing that the event is predicted to occur over the segment time range. Points represent the occurrence of non-interesting events. For a positive prediction (this is, the event is expected to happen), no indication is given about the specific time of the potential appearance of the predicted event; instead, the event would appear at any time of the prediction window if the forecast is positive; it might even appear more than once. Hence, segments on the right hand side of the vertical line represent current positive predictions of events, whereas past events and predictions appear on the left of the timeline. The outcome of each prediction is visualized as follows:

- True Positive: The event was predicted to happen and it does - for the remaining prediction window this is marked as True Positive - it is signalled with a small circle over the bar that marks the prediction (on times: -3.2 minutes in top event timeline #1, -0.4 minutes on the second timeline #2, 0 minutes on the fifth one, #3).
- False Positive: The event was predicted to happen and it does not - this can only be known at the end of the prediction window and it renders a False

Positive. This appears as a prediction window that finishes without the actual appearance of the event; it is marked with a cross (see the end of the third event timeline from the top, #4).

- False Negative: The event was not predicted to happen but it does happen. Whenever a critical or major event appears without a prediction bar, it is labelled with an isolated cross, such as the one at -2 minutes (#5).
- True Negative: The event was not predicted to happen and it does not - which can only be ensured at the end of the prediction window. Nothing is shown to avoid visual clutter.

The complete system was deployed, both training and prediction subsystems, in two different settings: a laptop equipped with an Intel i7-3720QM 2.6GHz processor, 8 virtual cores with HyperThreading, 16GB of RAM running Ubuntu x64 15.10 as operating system; and a hardware cluster composed by the master machine and seven workers - each node an HP ProLiant SL210t G8 with 2 Intel Xeon processors E5-2630v2 (2.6 GHz), 6 cores each, for 12 physical cores (up to 24 logical cores with HyperThreading); 32 GB of RAM and 3 TB SATA storage.

One important factor to take into account is how much memory the whole system takes. I measured the memory requirements of the trained models, and found an upper limit of 4.4 MB per model, rendering the total memory size for a production system at most at $108 * 4.4 = 475.2$ MB. Although additional memory is required for the observation and prediction windows, this figure is fairly low. At less than 10 MB per model and considering current storage prices, the system can be considered pretty lightweight.

Another limitation to consider is prediction lag: The prediction process inserts a delay between the end of the observation window and the start of the prediction window. This delay cannot be higher than the prediction update period. Otherwise, saturation would happen. In my experiments on both deployment settings, predicting the occurrence of all 108 events of interest took around 350 milliseconds in the portable computer, and 330 in the cluster. Each prediction requires

traversing all trees in the model, rounding up to $treesPerModel * numberOfModels = 50 * 108 = 5400$ decision trees. Trees have a variable number of depth levels, as 9 is just each tree's depth upper bound. This suggests that Spark's scheduling algorithms are working far from the saturation level. Both deployments show an adequate latency and capacity for the problem at hand, and seem to provide a valid basis for more demanding environments. While there is a whole set of variables that influence the exact performance of the prediction phase, such as the number of event types of interest, the length of the prediction window and the time pace new prediction windows are opened, this result, combined with the previous paragraph's result shows that an Online Failure Prediction architecture based on Spark is, not only feasible, but can even be deployed on commodity hardware with good results.

8.1.3 CONTRIBUTION SUMMARY

In summary, I have shown how to build a proactive fault management system based on open source, Big Data technologies such as Spark, HDFS and Kafka. As I talked about at the beginning of this chapter, the results I have shown are not only important because they show a practical implementation of the basic Online Failure Prediction concepts we've been seeing along the thesis, but because they show how easy it is to deploy them with no special hardware required. The usage of Spark's libraries greatly reduces the complexity of the tasks to accomplish: Spark MLlib provides parallelized implementations of Machine Learning algorithms and Spark Streaming manages online computations effortlessly.

Regarding future work that can be spun from this research, the most obvious line of work is adding online learning capabilities to the Training Subsystem. Training models, specially Random Forests, is computationally expensive, so adding ways to avoid it and make the models keep up with system drifts would be an optimal solution. Another clear possible work is to, simply, test the architecture and proposal on different scenarios to stress test the system: increasing the events in the windows, the events to predict and the prediction frequency.

9

Conclusions

This chapter closes the work contained in this book. On it, I will give an overview of everything I've talked about to give a general image of my research. My thesis addressed the problem of managing failures on distributed systems. Specifically, tasks related to predicting the occurrence of a failure and establishing its cause. These tasks are become more and more important as time goes by, due to the growth and prevalence of distributed systems of all type on different scenarios. Alongside this growth, their economic importance also goes up: a server downtime can cost up to half a million dollars per hour to certain companies. As they grow in size and importance, they get harder to manage, as there is more data to monitor and less reaction time. In fact, ideally, there should be no reaction time to failures as they should be anticipated and measures should be taken to avoid them. Under this situation, I exposed how scientific progress on this topic is ample but scattered. Most research is done isolated, with no way to compare different pro-

posals. Furthermore, there is a set of open problems that arise when defining the requirements of future failure management systems, as they require to have minimum human intervention, several tasks that are normally done by human experts must be completed by software systems on their own. These are two: the definition of the Observation Window, this is, the time period in the past to observe in order to predict future events, and studying the data in order to gather information about probable causes of the different errors. What is more, there is also no software architecture proposal that brings the failure prediction process to reality utilizing standard Big Data components, which allow it to be plugged to most distributed systems with minimal hassle.

Facing the research landscape and open problems I talked about on the previous paragraph, I proposed four different contributions that attempt to solve them and create a more cohesive research situation. I did so utilizing two real failure datasets, one obtained from a Computer Network from a big Spanish bank and the other one coming from the Los Alamos National Laboratory, detailing failures on a High Performance Computing Cluster.

First, to allow for algorithm comparison I tested a set of Machine Learning algorithms against both scenarios, analyzing their parameters' behavior and their overall performance. Results pointed to Random Forests and Support Vector Machines as the best performing algorithms of the tested set, with similar scores. They were quite different, though, in terms of the influence of their parameters: while Random Forests is quite robust and is not heavily affected by most of its parameters, being the number of trees to grow the only one that acts as a "performance threshold", Support Vector Machines are highly dependant on both of their parameters to perform adequately, requiring thorough tuning to do so. Based on this conclusion, I chose to base my next contributions on Random Forests, as it would allow me to avoid performing tuning for each grown model. This part covered my first objective.

After the model comparison I moved to the problem of choosing an optimal Observation Window, fulfilling objective number two. This is one of the problems that requires human expertise to accomplish. In previous research papers, most au-

thors either ignored it or just performed a grid search. But that is not enough. I propose to use a multiple Observation Window scheme, that is much more flexible, as it permits the model to capture different relevant time periods without including noise. And to avoid needing an expert, I use a customized Genetic Algorithm, that automatically chooses the best combination of windows. It is also compensated with the total window size, forcing the algorithm to find a trade off between raw model performance and window size. This can also be customized depending on the importance of storage needs on a certain scenario. My experiments, again, on both scenarios, show that my proposal outperforms simple grid searches most of the time and the multiple window scheme is employed several times to find complex solutions that would be impossible to find just by brute force.

Regarding the third contribution and objective, it deals with the problem of analyzing the causes of errors on a system. The current standard for this is a technique called Bayesian Networks, which, again, requires previous knowledge about the system to be created. In contrast to it, my proposal is a data structure that leverages the concept of influence of a feature in a prediction model. While the exact mathematical foundation of it depends on the specific algorithm used, the intuition behind it is the same: it is a number that measures how much the appearance of a certain event influenced the appearance of the target event. Using this, I create an Influence Matrix, containing the influence of every event on each other. Applying it to the Networking scenario, I show how it is a compact, highly summarized data structure that, at the same time, reveals a lot of information about the modelled system. There are several very interesting facts that can be extracted just by observing it: the system topology, several metrics and indices that signal dangerous events and even correlation chains that go from harmful events to critical events, allowing a network operator to have an unprecedented image of how the system's parts influence each other.

The last contribution of this thesis is an instantiation of the Failure Prediction process, as well as a validation of the main concept of it by showing that it can be done utilizing industry-ready, open source software, which covers my fourth and last objective. It details a Big Data Online Failure Prediction architecture.

It is built utilizing Spark and two of its libraries: Spark MLlib, to train Random Forests models, and Spark Streaming, to perform the online processing. Additionally, it utilizes Kafka as a connector to the managed system, so it is highly flexible communication-wise. Apart from its theoretical design, we implemented it on a big data cluster, showing that Spark MLlib's implementation of Random Forests performs adequately and how, even when running on a personal laptop, hundreds of models can perform in an online fashion with minimal memory requirements.

All in all, this thesis summarizes my effort in bridging the gap between academic Failure Management research and industrial situations, as this scientific discipline is one that is directly applicable to practical situations, assuming the implementation issues are addressed.

9.1 METHODOLOGICAL RISKS AND LIMITATIONS

Taking the contributions I just exposed, I consider the objectives I detailed on chapter 3 to be fulfilled. Nevertheless, there are certain aspects I must remark to frame the project as faithfully as possible, as there are certain limitations to it. I'll go over every one that I've found, explain it and, if applicable, state how I've tried to solve or mitigate it.

1. **Dataset related risks:** every research that draws conclusion from analysis performed on data must acknowledge the limitations and potential risks this poses. In my case, these are: first, possible lack of enough samples. As some of my contributions require generating prediction models for every event on the dataset, some of these models may be incorrect or generated using noise if there is not a large enough number of samples. I try to avoid this problem by performing cross-validation to check each model's performance and set certain performance thresholds to avoid including bad performing models in my calculations. Second, each dataset has its own features, its own variances, its own distributions, its own examples, etc. So, there is always the risk of finding certain patterns that are not general. While

my thesis certainly suffers from this problem, I don't think it influences or harms its results, as most of the contributions propose methodologies not directly related to the model's performance and, while the first contribution's results are models' performances, I evaluate each Machine Learning algorithm against ten different events in order to avoid any kind of problem with certain single events.

2. **Reproducibility:** regarding the reproducibility of my results, this is, the capability of obtaining the same results if the same analysis are carried out independently with the same data, I consider it to be totally reproducible. The great majority of the analysis have been ran using the R programming language, so if anyone wanted to reproduce my results, they can just download one of the datasets that I used (the LANL dataset is public, while our networking data is protected by an NDA) and apply the algorithms and methods I propose, which are published on [111]. Results should be very close if not equal.
3. **Replicability:** on the other side, this is a more dangerous risk than reproducibility. The consistency of my methods producing good results is something that I can only assess on the datasets that I, personally, test. Of course, there is nothing that would lead me to think that they wouldn't perform correctly in other environments or using other datasets, but it is still, nevertheless, a potential risk of any research like this.
4. **Model training assumptions:** while not a methodological risk per se, this is something I have to make clear. As I've stated, one of my main objectives was to reduce the need for human interaction in the Failure Management process. This includes the process of designing the best possible Machine Learning models. As we saw when I explored my datasets that they present an unstable behavior, changing it as time goes by. Thus, a deeper, more complex preprocessing that included this kind of variability as a model feature would, certainly, improve the models' performance. But this requires a

human expert performing data analysis. So, in order to show which kind of results can be obtained with just basic knowledge and adopting standard procedures, I avoided including this kind of information on my models. Nevertheless, results were still very good.

5. **Observation Window definition function:** as we saw on Chapter 4, I defined a specific windowing function for my datasets in order to be able to feed them to the Machine Learning algorithms. This is also a bias on my research, as there is no guarantee that that is an optimal transformation. Nevertheless, it is a standard one, so I consider the decision to be justified.
6. **Validation:** how to evaluate a contribution (or an artifact) is a crucial decision to take. For some chapters (Contributions 1 and 2), there is a clear way of doing so, using standard Data Science metrics. But with Contributions 3 and 4, things are a bit more murkier. Contribution 3 deals with a novel, automatic way of extracting knowledge from trained models. As such, the validation process is showing a set of different information and ideas that can be easily gained from it. Additionally, its results were validated by human experts. On its side, Contribution 4 is a software architecture proposal. It was validated by running tests and showing that every step was feasible and efficient. I consider both validation processes complete and enough, however, as they are not totally standard, I consider it appropriate to include them here as potential risks.

9.2 FUTURE WORK

As it happens with virtually any research project, there are still things to be done and lines of work that this thesis opens. The most immediate one is to replicate my results: testing the different proposals and methodologies on different and varied scenarios would strengthen them and confirm them as promising solutions for the problems they solve. Of course, there is an area of Failure Management that this work does not deal with: system healing, the actions to take to restore the system to

its normal state when a failure has happened, was not treated at all on my research, as the scenarios we had did not have enough information to do so. Thus, an expansion of the same principles I've exposed to work with system healing would improve the completeness of my proposal. Lastly, there is one feature of the ideal solutions that I outlined on Chapter 3 that my solutions did not have: updatability. Due to the base model I employed, Random Forests, the trained models are only valid as long as the system does not suffer any relevant change in its configuration or topology. A completely autonomous Failure Management system should update its models on its own as the system changes. I see two possible solutions for this issue: one, to use prediction models that are inherently updatable, such as k-nearest neighbours. This is the most straightforward one, but it forces the users to use a subset of all the available models. The other potential solution requires more work but allows for much more flexibility. It would be based on establishing a model performance checking routine, that updated the models' performance scores as time goes by. Then, it could force them to retrain if the obtained scores drifted away too much from the original one. Performing this process in parallel or at non-important times would allow the system to update itself in real time without needing to be unplugged.

References

- [1] The data science venn diagram. <http://drewconway.com/zia/2013/3/26/the-data-science-venn-diagram>. Accessed: 2018-05-04.
- [2] Fcaps. <http://www.tech-faq.com/fcaps.html>. Accessed: 2018-05-13.
- [3] Andrew S Tanenbaum and Maarten Van Steen. *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.
- [4] Downtime, outages and failures - understanding their true costs. <https://www.evolver.com/blog/downtime-outages-and-failures-understanding-their-true-costs.html>. Accessed: 2018-05-13.
- [5] R Hevner Von Alan, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, 28(1):75–105, 2004.
- [6] John W Tukey. The future of data analysis. *The annals of mathematical statistics*, 33(1):1–67, 1962.
- [7] John W Tukey. *Exploratory data analysis*, volume 2. Reading, Mass., 1977.
- [8] Peter Naur. *Concise survey of computer methods*. Studentlitteratur, 1974.
- [9] Nina Zumel, John Mount, and Jim Porzak. *Practical data science with R*. Manning, 2014.
- [10] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2014.
- [11] José M Navarro, G Hugo A Parada, and Juan C Dueñas. System failure prediction through rare-events elastic-net logistic regression. In *Artificial Intelligence, Modelling and Simulation (AIMS), 2014 2nd International Conference on*, pages 120–125. IEEE, 2014.

- [12] José M Navarro, Juan C Dueñas, et al. Classification in sparse, high dimensional environments applied to distributed systems failure prediction. In *International Conference on Artificial Intelligence and Soft Computing*, pages 714–726. Springer, 2015.
- [13] Jose Manuel Navarro Gonzalez, Javier Andion Jimenez, and Juan Carlos Duenas Lopez. Optimizing failure prediction time windows through evolutionary computation. *IEEE Computational Intelligence Magazine*, Under Review.
- [14] Jose Manuel Navarro Gonzalez, Javier Andion Jimenez, Juan Carlos Duenas Lopez, et al. Root cause analysis of network failures using machine learning and summarization techniques. *IEEE Communications Magazine*, 55(9):126–131, 2017.
- [15] Juan C Duenas, Jose M Navarro, Javier Andion, Felix Cuadrado, et al. Applying event stream processing to network online failure prediction. *IEEE Communications Magazine*, 56(1):166–170, 2018.
- [16] Javier Andión Jiménez, José Manuel Navarro González, Manuel Álvarez Campana Fernández-Corredor, and Juan Carlos Dueñas López. A passive, non-intrusive, cheap method to identify behaviours and habits in the campus. In *XIII Jornadas de Ingenieria Telematica - JITEL2017*, pages 40–47. Universitat Politècnica Valencia, September 2017.
- [17] Felix Salfner, Maren Lenk, and Miroslaw Malek. A survey of online failure prediction methods. *ACM Comput. Surv.*, 42(3):10:1–10:42, March 2010.
- [18] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. Network anomaly detection: Methods, systems and tools. *IEEE Communications Surveys Tutorials*, 16(1):303–336, First 2014.
- [19] A. L. Buczak and E. Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys Tutorials*, 18(2):1153–1176, Secondquarter 2016.
- [20] F. Zaman, G. Hogan, S. V. Der Meer, J. Keeney, S. Robitzsch, and G. m. Muntean. A recommender system architecture for predictive telecom network management. *IEEE Communications Magazine*, 53(1):286–293, January 2015.

- [21] Ranganai Chaparadza. Unifaff: a unified framework for implementing autonomic fault management and failure detection for self-managing networks. *International Journal of Network Management*, 19(4):271–290, 2009.
- [22] X. Fu, R. Ren, S. A. McKee, J. Zhan, and N. Sun. Digging deeper into cluster system logs for failure prediction and root cause diagnosis. In *2014 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 103–112, Sept 2014.
- [23] Masataka Sonoda, Yukihiro Watanabe, and Yasuhide Matsumoto. Prediction of failure occurrence time based on system log message pattern learning. In *Network Operations and Management Symposium (NOMS)*, 2012 IEEE, pages 578–581. IEEE, 2012.
- [24] Tatsuaki Kimura, Akio Watanabe, Tsuyoshi Toyono, and Keisuke Ishibashi. Proactive failure detection learning generation patterns of large-scale network logs. In *Network and Service Management (CNSM)*, 2015 11th International Conference on, pages 8–14. IEEE, 2015.
- [25] Jiang Zhong, Weili Guo, and Zhenhua Wang. Study on network failure prediction based on alarm logs. In *Big Data and Smart City (ICBDSC)*, 2016 3rd MEC International Conference on, pages 1–7. IEEE, 2016.
- [26] Felix Salfner and Mirosław Malek. Using hidden semi-markov models for effective online failure prediction. In *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on*, pages 161–174. IEEE, 2007.
- [27] Dapeng Liu, Youjian Zhao, Haowen Xu, Yongqian Sun, Dan Pei, Jiao Luo, Xiaowei Jing, and Mei Feng. Opprentice: Towards practical and automatic anomaly detection through machine learning. In *Proceedings of the 2015 Internet Measurement Conference*, pages 211–224. ACM, 2015.
- [28] Nentawe Gurumdimma, Arshad Jhumka, Maria Liakata, Edward Chuah, and James Browne. Towards detecting patterns in failure logs of large-scale distributed systems. In *Parallel and Distributed Processing Symposium Workshop (IPDPSW)*, 2015 IEEE International, pages 1052–1061. IEEE, 2015.
- [29] Q. Guan and S. Fu. Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures. In *2013 IEEE 32nd International Symposium on Reliable Distributed Systems*, pages 205–214, Sept 2013.

- [30] Meimei Ding and Hui Tian. Pca-based network traffic anomaly detection. *Tsinghua Science and Technology*, 21(5):500–509, 2016.
- [31] Kenji Yamanishi and Yuko Maruyama. Dynamic syslog mining for network failure monitoring. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 499–508. ACM, 2005.
- [32] Shunsuke Hirose, Kenji Yamanishi, Takayuki Nakata, and Ryohei Fujimaki. Network anomaly detection based on eigen equation compression. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1185–1194. ACM, 2009.
- [33] T. Chalermarrewong, T. Achalakul, and S. C. W. See. Failure prediction of data centers using time series and fault tree analysis. In *2012 IEEE 18th International Conference on Parallel and Distributed Systems*, pages 794–799, Dec 2012.
- [34] Yukihiro Watanabe, Hiroshi Otsuka, Masataka Sonoda, Shinji Kikuchi, and Yasuhide Matsumoto. Online failure prediction in cloud datacenters by real-time message pattern learning. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pages 504–511. IEEE, 2012.
- [35] Xiaoyu Fu, Rui Ren, Jianfeng Zhan, Wei Zhou, Zhen Jia, and Gang Lu. Log-master: mining event correlations in logs of large-scale cluster systems. In *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on*, pages 71–80. IEEE, 2012.
- [36] Q. Guan and S. Fu. auto-aid: A data mining framework for autonomic anomaly identification in networked computer systems. In *International Performance Computing and Communications Conference*, pages 73–80, Dec 2010.
- [37] Qiang Guan, Ziming Zhang, and Song Fu. Proactive failure management by integrated unsupervised and semi-supervised learning for dependable cloud systems. In *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*, pages 83–90. IEEE, 2011.
- [38] Edward Chuah, Arshad Jhumka, Sai Narasimhamurthy, John Hammond, James C Browne, and Bill Barth. Linking resource usage anomalies

- with system failures from cluster log data. In *Reliable Distributed Systems (SRDS), 2013 IEEE 32nd International Symposium on*, pages 111–120. IEEE, 2013.
- [39] Alejandro Pelaez, Andres Quiroz, James C Browne, Edward Chuah, and Manish Parashar. Online failure prediction for hpc resources using decentralized clustering. In *High Performance Computing (HiPC), 2014 21st International Conference on*, pages 1–9. IEEE, 2014.
 - [40] Z. Zheng, Z. Lan, R. Gupta, S. Coghlan, and P. Beckman. A practical failure prediction with location and lead time for blue gene/p. In *2010 International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 15–22, June 2010.
 - [41] Yinglung Liang, Yanyong Zhang, A Sivasubramaniam, Morris Jette, and R Sahoo. Bluegene/l failure analysis and prediction models. In *DSN '06 Proceedings of the International Conference on Dependable Systems and Networks*, volume 2006, pages 425–434, 02 2006.
 - [42] Li Yu, Ziming Zheng, Zhiling Lan, and Susan Coghlan. Practical online failure prediction for blue gene/p: Period-based vs event-driven. In *Dependable Systems and Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on*, pages 259–264. IEEE, 2011.
 - [43] Jungang Xu and Hui Li. The failure prediction of cluster systems based on system logs. In *International Conference on Knowledge Science, Engineering and Management*, pages 526–537. Springer, 2013.
 - [44] Ziming Zheng, Zhiling Lan, Byung H Park, and Al Geist. System log pre-processing to improve failure prediction. In *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on*, pages 572–577. IEEE, 2009.
 - [45] B. Schroeder and G. Gibson. A large-scale study of failures in high-performance computing systems. *IEEE Transactions on Dependable and Secure Computing*, 7(4):337–350, Oct 2010.
 - [46] Hongbing Wang, Lei Wang, Qi Yu, Zibin Zheng, Athman Bouguettaya, and Michael R Lyu. Online reliability prediction via motifs-based dynamic bayesian networks for service-oriented systems. *IEEE Transactions on Software Engineering*, 43(6):556–579, 2017.

- [47] Ilenia Fronza, Alberto Sillitti, Giancarlo Succi, Mikko Terho, and Jelena Vlasenko. Failure prediction based on log files using random indexing and support vector machines. *Journal of Systems and Software*, 86(1):2–11, 2013.
- [48] Sandia National Laboratories, United States. National Nuclear Security Administration, United States. Department of Energy. Office of Scientific, and Technical Information. *High-end System Event Correlations and Predictions*. United States. National Nuclear Security Administration, 2014.
- [49] Catello Di Martino. *One Size Does Not Fit All: Clustering Supercomputer Failures Using a Multiple Time Window Approach*, pages 302–316. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
- [50] I. G. Cârjeu, T. Shorrock, and M. Seeger. Clustering it events around common root causes. In *2014 IEEE International Conference on Services Computing*, pages 749–757, June 2014.
- [51] Bingpeng Zhu, Gang Wang, Xiaoguang Liu, Dianming Hu, Sheng Lin, and Jingwei Ma. Proactive drive failure prediction for large scale storage systems. In *Mass Storage Systems and Technologies (MSST), 2013 IEEE 29th Symposium on*, pages 1–5. IEEE, 2013.
- [52] J. Li, X. Ji, Y. Jia, B. Zhu, G. Wang, Z. Li, and X. Liu. Hard drive failure prediction using classification and regression trees. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 383–394, June 2014.
- [53] L. P. Queiroz, F. C. M. Rodrigues, J. P. P. Gomes, F. T. Brito, I. C. Chaves, M. R. P. Paula, M. R. Salvador, and J. C. Machado. A fault detection method for hard disk drives based on mixture of gaussians and nonparametric statistics. *IEEE Transactions on Industrial Informatics*, 13(2):542–550, April 2017.
- [54] I. Irrera and M. Vieira. A practical approach for generating failure data for assessing and comparing failure prediction algorithms. In *2014 IEEE 20th Pacific Rim International Symposium on Dependable Computing*, pages 86–95, Nov 2014.

- [55] T. Pitakrat, D. Okanovic, A. V. Hoorn, and L. Grunske. An architecture-aware approach to hierarchical online failure prediction. In *2016 12th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA)*, pages 60–69, April 2016.
- [56] Cheng Zhou, Boris Cule, and Bart Goethals. A pattern based predictor for event streams. *Expert Systems with Applications*, 42(23):9294 – 9306, 2015.
- [57] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han. Outlier detection for temporal data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 26(9):2250–2267, Sept 2014.
- [58] C. Wang, H. T. Vo, and P. Ni. An iot application for fault diagnosis and prediction. In *2015 IEEE International Conference on Data Science and Data Intensive Systems*, pages 726–731, Dec 2015.
- [59] H. Li, B. Qian, D. Parikh, and A. Hampapur. Alarm prediction in large-scale sensor networks - a case study in railroad. In *2013 IEEE International Conference on Big Data*, pages 7–14, Oct 2013.
- [60] L. Bennacer, Y. Amirat, A. Chibani, A. Mellouk, and L. Ciavaglia. Self-diagnosis technique for virtual private networks combining bayesian networks and case-based reasoning. *IEEE Transactions on Automation Science and Engineering*, 12(1):354–366, Jan 2015.
- [61] L. Bennacer, L. Ciavaglia, S. Ghamri-Doudane, A. Chibani, Y. Amirat, and A. Mellouk. Scalable and fast root cause analysis using inter cluster inference. In *2013 IEEE International Conference on Communications (ICC)*, pages 3563–3568, June 2013.
- [62] T. Kimura, K. Ishibashi, T. Mori, H. Sawada, T. Toyono, K. Nishimatsu, A. Watanabe, A. Shimoda, and K. Shiimoto. Spatio-temporal factorization of log data for understanding network events. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 610–618, April 2014.
- [63] J. Li, C. Liu, J. X. Yu, Y. Chen, T. Sellis, and J. S. Culpepper. Personalized influential topic search via social network summarization. *IEEE Transactions on Knowledge and Data Engineering*, 28(7):1820–1834, July 2016.
- [64] Ren Quinn, Josh Kunz, Aisha Syed, Joe Breen, Sneha Kasera, Rob Ricci, and Jacobus Van der Merwe. Knownet: Towards a knowledge plane for

- enterprise network management. In *Network Operations and Management Symposium (NOMS)*, 2016 IEEE/IFIP, pages 249–256. IEEE, 2016.
- [65] William Q Meeker and Yili Hong. Reliability meets big data: opportunities and challenges. *Quality Engineering*, 26(1):102–116, 2014.
 - [66] Jay Lee, Behrad Bagheri, and Hung-An Kao. Recent advances and trends of cyber-physical systems and big data analytics in industrial informatics. In *International proceeding of int conference on industrial informatics (INDIN)*, pages 1–6, 2014.
 - [67] Jay Lee, Edzel Lapira, Behrad Bagheri, and Hung-an Kao. Recent advances and trends in predictive manufacturing systems in big data environment. *Manufacturing Letters*, 1(1):38–41, 2013.
 - [68] Ibrahim Abaker Targio Hashem, Ibrar Yaqoob, Nor Badrul Anuar, Salimah Mokhtar, Abdullah Gani, and Samee Ullah Khan. The rise of “big data” on cloud computing: Review and open research issues. *Information Systems*, 47:98–115, 2015.
 - [69] Min Chen, Shiwen Mao, and Yunhao Liu. Big data: A survey. *Mobile networks and applications*, 19(2):171–209, 2014.
 - [70] CL Philip Chen and Chun-Yang Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information Sciences*, 275:314–347, 2014.
 - [71] Stephen Kaisler, Frank Armour, J Alberto Espinosa, and William Money. Big data: Issues and challenges moving forward. In *System sciences (HICSS)*, 2013 46th Hawaii international conference on, pages 995–1004. IEEE, 2013.
 - [72] Karthik Kambatla, Giorgos Kollias, Vipin Kumar, and Ananth Grama. Trends in big data analytics. *Journal of Parallel and Distributed Computing*, 74(7):2561–2573, 2014.
 - [73] Dmitry Namiot. On big data stream processing. *International Journal of Open Information Technologies*, 3(8), 2015.
 - [74] Han Hu, Yonggang Wen, Tat-Seng Chua, and Xuelong Li. Toward scalable systems for big data analytics: A technology tutorial. *IEEE access*, 2:652–687, 2014.

- [75] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy Mccauley, M Franklin, Scott Shenker, and Ion Stoica. Fast and interactive analytics over hadoop data with spark. *Usenix Login*, 37(4):45–51, 2012.
- [76] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy Mccauley, M Franklin, Scott Shenker, and Ion Stoica. Fast and interactive analytics over hadoop data with spark. *Usenix Login*, 37(4):45–51, 2012.
- [77] R Shyam, Sachin Kumar, Prabakaran Poornachandran, and KP Soman. Apache spark a big data analytics platform for smart grid. *Procedia Technology*, 21:171–178, 2015.
- [78] Aaron N Richter, Taghi M Khoshgoftaar, Sara Landset, and Tawfiq Hasanin. A multi-dimensional comparison of toolkits for machine learning with big data. In *Information Reuse and Integration (IRI)*, 2015 *IEEE International Conference on*, pages 1–8. IEEE, 2015.
- [79] Sara Landset, Taghi M Khoshgoftaar, Aaron N Richter, and Tawfiq Hasanin. A survey of open source tools for machine learning with big data in the hadoop ecosystem. *Journal of Big Data*, 2(1):24, 2015.
- [80] Tyler Akidau, Alex Balikov, Kaya Bekiroğlu, Slava Chernyak, Josh Haberman, Reuven Lax, Sam McVeety, Daniel Mills, Paul Nordstrom, and Sam Whittle. Millwheel: fault-tolerant stream processing at internet scale. *Proceedings of the VLDB Endowment*, 6(11):1033–1044, 2013.
- [81] Mohiuddin Solaimani, Mohammed Iftexhar, Latifur Khan, Bhavani Thuraisingham, and Joey Burton Ingram. Spark-based anomaly detection over multi-source vmware performance data in real-time. In *Computational Intelligence in Cyber Security (CICS)*, 2014 *IEEE Symposium on*, pages 1–8. IEEE, 2014.
- [82] Tom Michael Mitchell. *The discipline of machine learning*, volume 9. Carnegie Mellon University, School of Computer Science, Machine Learning Department, 2006.
- [83] Taiwo Oladipupo Ayodele. Types of machine learning algorithms. In *New advances in machine learning*. InTech, 2010.

- [84] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.
- [85] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *Icml*, volume 96, pages 148–156. Bari, Italy, 1996.
- [86] S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.
- [87] M Shafie-Khah, M Parsa Moghaddam, and MK Sheikh-El-Eslami. Price forecasting of day-ahead electricity markets using a hybrid forecast method. *Energy Conversion and Management*, 52(5):2165–2169, 2011.
- [88] Hui Song, A Kai Qin, and Flora D Salim. Multivariate electricity consumption prediction with extreme learning machine. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 2313–2320. IEEE, 2016.
- [89] Ali Azadeh, SF Ghaderi, S Tarverdian, and Morteza Saberi. Integration of artificial neural networks and genetic algorithm to predict electrical energy consumption. *Applied Mathematics and Computation*, 186(2):1731–1741, 2007.
- [90] Liang Tian and Afzel Noore. On-line prediction of software reliability using an evolutionary connectionist model. *Journal of Systems and Software*, 77(2):173–180, 2005.
- [91] Chunkai Zhang and Hong Hu. Using pso algorithm to evolve an optimum input subset for a svm in time series forecasting. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, volume 4, pages 3793–3796. IEEE, 2005.
- [92] Esmail Hadavandi, Hassan Shavandi, and Arash Ghanbari. Integration of genetic fuzzy systems and artificial neural networks for stock price forecasting. *Knowledge-Based Systems*, 23(8):800–808, 2010.
- [93] Frauke Friedrichs and Christian Igel. Evolutionary tuning of multiple svm parameters. *Neurocomputing*, 64:107–117, 2005.

- [94] Frank Hung-Fat Leung, Hak-Keung Lam, Sai-Ho Ling, and Peter Kwong-Shun Tam. Tuning of the structure and parameters of a neural network using an improved genetic algorithm. *IEEE Transactions on Neural networks*, 14(1):79–88, 2003.
- [95] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy Mccauley, M Franklin, Scott Shenker, and Ion Stoica. Fast and interactive analytics over hadoop data with spark. *Usenix Login*, 37(4):45–51, 2012.
- [96] Salman Salloum, Ruslan Dautov, Xiaojun Chen, Patrick Xiaogang Peng, and Joshua Zhexue Huang. Big data analytics on apache spark. *International Journal of Data Science and Analytics*, 1(3-4):145–164, 2016.
- [97] Tom White. *Hadoop: The definitive guide*. ” O’Reilly Media, Inc.”, 2012.
- [98] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, 2016.
- [99] Sara Landset, Taghi M Khoshgoftaar, Aaron N Richter, and Tawfiq Hasanin. A survey of open source tools for machine learning with big data in the hadoop ecosystem. *Journal of Big Data*, 2(1):24, 2015.
- [100] Spark streaming. <https://spark.apache.org/streaming/>. Accessed: 2018-05-11.
- [101] Apache kafka. <https://kafka.apache.org/>. Accessed: 2018-05-11.
- [102] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.
- [103] Max Kuhn and Kjell Johnson. *Applied predictive modeling*, volume 26. Springer, 2013.
- [104] John W Tukey. Comparing individual means in the analysis of variance. *Biometrics*, pages 99–114, 1949.
- [105] Variable importance. <https://topepo.github.io/caret/variable-importance.html>. Accessed: 2018-05-04.

- [106] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.
- [107] Matei Zaharia, Tathagata Das, Haoyuan Li, Scott Shenker, and Ion Stoica. Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters. *HotCloud*, 12:10–10, 2012.
- [108] Khamisi Kalegele, Kazuto Sasai, Hideyuki Takahashi, Gen Kitagata, and Tetsuo Kinoshita. Four decades of data mining in network and systems management. *IEEE Transactions on Knowledge and Data Engineering*, 27(10):2700–2716, 2015.
- [109] Jiang Zheng and Aldo Dagnino. An initial study of predictive machine learning analytics on large volumes of historical data for power system applications. In *Big Data (Big Data), 2014 IEEE International Conference on*, pages 952–959. IEEE, 2014.
- [110] Ke Sun, Wansheng Miao, Xin Zhang, and Ruonan Rao. An improvement to feature selection of random forests on spark. In *Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on*, pages 774–779. IEEE, 2014.
- [111] Ofp-thesis. <https://github.com/josemanuelnavarro/OFP-Thesis>. Accessed: 2018-05-14.

Curriculum Vitae

José Manuel Navarro

May 2018

1 Academic Education

Telecommunications Engineering, Universidad Miguel Hernández de Elche, 2013.

2 Other Education

- "Experimental Research Statistics", Instituto de Ciencias de la Educación de la Universidad Politécnica de Madrid. 14 hours. 2017.
- "Technical and Scientific Documents Writing on LaTeX", Instituto de Ciencias de la Educación de la Universidad Politécnica de Madrid. 12 hours. 2017.
- "The Data Scientist's Toolbox", John Hopkins University through Coursera (online course). 1 month. 2015.
- "R Programming", John Hopkins University through Coursera (online course). 1 month. 2015.
- "Getting and Cleaning Data", John Hopkins University through Coursera (online course). 1 month. 2015.
- "Lisbon Machine Learning School", Instituto Superior Técnico do Lisboa. 7 days. 2014.

3 Funded Research Projects Participation

3.0.1 M2C2

- Funding: National R+D Plan.
- Participants: UPM, UPV, U. Cantabria.
- Duration: 1/1/2016-31/12/2018. 36 months.
- Partial dedication.

3.0.2 Creation of a biomedical technologies observatory

- Funding: Instituto de Salud Carlos III.
- Duration: 1/9/2016-31/12/2016. 4 months.
- Main researcher: Gonzalo León.
- Partial dedication.

3.0.3 Development of an anomaly detector for wind turbines through Machine Learning

- Funding: Fujitsu Spain.
- Duration: 1/4/2017-31/7/2017. 3 months.
- Main researcher: Juan Carlos Dueñas.
- Partial dedication.

3.0.4 L3P3: Online Failure Prediction

- Funding: Article 83 contract.
- Duration: 1/9/2013-1/9/2016. 3 years.
- 5 researchers team.
- Technical director.

4 Research articles

4.0.1 Optimizing Failure Prediction Time Windows Through Evolutionary Computation

- Authors: José M. Navarro, Javier Andión y Juan C. Dueñas.
- Publication: IEEE Communications Magazine.
- Date: Under review.

4.0.2 Applying Events Stream Processing to Network Online Failure Prediction

- Authors: Juan C. Dueñas, José M. Navarro, Hugo A. Parada G., Javier Andión, Félix Cuadrado.
- Publication: IEEE Communications Magazine.
- Date: January 2018.

4.0.3 Root Cause Analysis of Network Failures using Machine Learning and Summarization Techniques

- Authors: José M. Navarro, Javier Andión, Juan C. Dueñas, Hugo A. Parada G.
- Publication: IEEE Communications Magazine.
- Date: September 2017.

5 Conference papers

5.0.1 A passive, non-intrusive, cheap method to identify behaviours and habits in the Campus

- Authors: Javier Andión, José M. Navarro, Manuel Álvarez-Campana, Juan C. Dueñas.
- Conference: XIII Jornadas de Ingeniería Telemática in Valencia, Spain.
- Date: September 2017.

5.0.2 Classification in sparse, high dimensional environments applied to distributed systems failure prediction

- Authors: Navarro, José M.; Parada, Hugo A.; Dueñas, Juan C.
- Conference: International Conference on Artificial Intelligence and Soft Computing in Zakopane, Poland.
- Date: September 2017.

5.0.3 System Failure Prediction through Rare-events Elastic-Net Logistic Regression

- Authors: Navarro, José M.; Parada, Hugo A.; Dueñas, Juan C..
- Conference: 2014 2nd International Conference on Artificial Intelligence, Modelling and Simulation in Madrid, Spain.
- Date: September 2014.

5.0.4 Efficient Wavelet Sign Prediction: Simulated Annealing vs Genetic Algorithms

- Authors: J. M. Navarro, P. Moreno, F. Rodríguez-Ballester, A. Martí, M. A. Cruz-Chávez, M. P. Malumbres, O. López.
- Conference: 16th International Conference on Knowledge-based and Intelligent Information and Engineering Systems in San Sebastián, Spain.
- Date: September 2012.

5.0.5 Analysis of genetic programming's parameters for 2D wavelet sign coding

- Authors: J. M. Navarro, O. López-Granado, M. P. Malumbres.
- Conference: II Workshop on Multimedia Data Coding and Transmission in Elche, Spain.
- Date: September 2012.

6 Invited talks, internships and other merits

1. 2015-2018: Teaching assistant at several Programming and Data Analysis subjects. Prepared and imparted two Data Science practical workshops on the Applications and Services subject of the Máster Universitario en Ingeniería de Telecomunicación of the ETSIT. Codirector of three Bachelor's Final Projects and one Master Thesis.
2. 2017: "Introduction to caret", talk given at the Machine Learning Spain Meetup of Madrid at Google Campus Madrid on April 2017.
3. 2016: Internship at the National Institute of Informatics of Tokyo, from February to May, working on the "Big Data for Disasters" project under Professor Yusheng Ji.
4. 2016-present: reviewer of the Journal of Software and Systems Magazine.
5. 2015:"Machine Learning and R", talk given at the Center for Open Middleware on June 2015.
6. 2015:"Introduction to R", talk given at the Center for Open Middleware on May 2015.
7. 2015:"Failure Prediction in Banks Networks using Machine Learning", talk given at the Machine Learning Spain Meetup of Madrid at Medialab Prado on April 2015.
8. 2015: "Data Science and Machine Learning workshop", talk given at the Escuela Superior de Telecomunicaciones of the UPM on March 2015.