MACHINE LEARNING TECHNIQUES APPLIED TO HYDRATE FAILURE DETECTION ON PRODUCTION LINES

Matheus Araújo Marins

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientadores: Sergio Lima Netto
Eduardo Antônio Barros da Silva

Rio de Janeiro
Setembro de 2018

MACHINE LEARNING TECHNIQUES APPLIED TO HYDRATE FAILURE DETECTION ON PRODUCTION LINES

Matheus Araújo Marins

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

Prof. Sergio Lima Netto, Ph.D.

Prof. Eduardo Antônio Barros da Silva, Ph.D.

Prof. Amaro Azevedo de Lima, Ph.D.

Eng. Thiago de Sá Feital, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2018

Marins, Matheus Araújo

Machine Learning Techniques Applied to Hydrate Failure Detection on Production Lines/Matheus Araújo Marins. – Rio de Janeiro: UFRJ/COPPE, 2018.

XV, 67 p.: il.; 29, 7cm.

Orientadores: Sergio Lima Netto

Eduardo Antônio Barros da Silva

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2018.

Referências Bibliográficas: p. 63 – 67.

1. machine learning. 2. random forest. 3. oil and gas. 4. hydrate failure. 5. data analysis. I. Netto, Sergio Lima *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

Agradecimentos

Certamente essa é a parte mais importante e difícil de escrever desse texto. Considero-me privilegiado por ter tido o apoio de tantas pessoas magníficas durante a minha jornada, a tornando muito mais simples e gratificante. A todas essas pessoas, meu muito obrigado, sou eternamente grato!

Agradeço à minha mãe e ao meu pai pelo apoio incondicional que me é oferecido, que faz com que cada conquista que eu obtive tenha mais valor. Me orgulho de ser filho de vocês e tento, sempre, absorver todo o ensinamento que vocês me passam para que um dia eu seja um pai a altura de vocês.

Agradeço ao meu irmão Felipe por ser o meu exemplo de honestidade, fidelidade e determinação. Obrigado por ter facilitado todo o caminho desde o Ensino Médio até aqui, e peço que continue me auxiliando; sua ajuda é indispensável! Agradeço também à minha cunhada Camila, que desde que entrou na nossa família tem sido essencial!

Ao meu orientador Sergio Lima Netto eu agradeço por ter aceitado encarar mais essa aventura ao meu lado. Muito obrigado pela paciência, zelo, piadas e eventuais broncas. Ter você como meu orientador me dá a certeza de estar seguindo um bom caminho e espero poder contar com você nos próximos capítulos.

Ao meu novo orientador Eduardo Antônio Barros da Silva, agradeço imensamente pela orientação prestada no mestrado; certamente o ganho foi muito significativo, apesar do pouco tempo.

Agradeço também aos demais professores e funcionários do SMT por ter me oferecido toda a estrutura necessária para a realização dessa dissertação. Agradeço especialmente ao professor Marcello Campos e à Dona Edinalva.

Aos meus amigos do SMT-2, em especial ao Felipe, Cinelli e Igor: muito obrigado por terem prestado o papel de "terceiro orientador"; cada resultado que será apresentado nesse trabalho tem uma contribuição indireta de vocês. Espero que essa amizade seja tão significativa para vocês quanto é para mim.

Agradeço aos meus amigos do Severo, e aproveito o espaço para me desculpar pela ausência nos últimos meses. Vocês significam muito para mim, cada um do grupo é especial.

Agradeço também à Coordenação de Aperfeiçoamento de Pessoal de Nível Supe-

rior (CAPES) e à Fundação Carlos Chagas Filho de Amparo à Pesquisa do Estado do Rio de Janeiro (FAPERJ) por me fornecerem suporte financeiro durante meu mestrado.

Agradeço aos prof. Amaro Azevedo Lima, prof. Sergio Lima Netto, prof. Eduardo Antônio Barros da Silva e ao pesquisador Thiago de Sá Feital por participarem da minha banca de mestrado.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

TÉCNICAS DE APRENDIZADO DE MÁQUINA APLICADAS NA DETECÇÃO DE HIDRATO EM LINHA DE PRODUÇÃO

Matheus Araújo Marins

Setembro/2018

Orientadores: Sergio Lima Netto
Eduardo Antônio Barros da Silva

Programa: Engenharia Elétrica

Este trabalho apresenta uma metodologia que cobre todo o desenvolvimento de um sistema de classificação de falhas relacionadas à formação de hidrato em linhas de produção de plataformas de petróleo. Serão utilizadas três bases de dados no desenvolvimento desse trabalho, onde cada uma delas é composta por uma variedade de medidas provenientes de sensores relacionados a poços. Nossa metodologia cobre todas as etapas de limpeza dessas bases: identificação de *tags* numéricas e categóricas; remoção de valores espúrios e de *outliers*; tratamento de dados faltantes através de interpolação; e a identificação de falhas e *tags* relevantes na plataforma.

Desenvolvemos um *framework* formado por diversas técnicas clássicas da área de Aprendizado de Máquina. O sistema proposto é composto por três grandes blocos: o primeiro irá extrair as características estatísticas de cada sinal de entrada através de uma janela deslizante; o segundo bloco irá mapear a saída do bloco anterior em um espaço mais apropriado através de duas tranformações: *z-score* e *Principal Components Analysis* (PCA); o último bloco é o classificador, que no caso optamos por ser o classificador *Random Forest*.

Também propomos uma técnica para aumentar a confiabilidade das amostras referentes ao estado de operação normal da plataforma. Quando lidamos com dados reais, é muito comum que muitas amostras estejam marcadas erradas, ou seja, os seus rótulos não refletem o estado real de operação da plataforma. Para suavizar esse efeito indesejado, desenvolvemos um método para remover amostras com marcações erradas, com o qual melhoramos a performance do modelo em 7,93%, na média, alcançando mais de 80% de acurácia em todos os cenários de classificação de uma única classe.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

MACHINE LEARNING TECHNIQUES APPLIED TO HYDRATE FAILURE DETECTION ON PRODUCTION LINES

Matheus Araújo Marins

September/2018

Advisors: Sergio Lima Netto
Eduardo Antônio Barros da Silva

Department: Electrical Engineering

The present work proposes a methodology that covers the whole process of classifying hydrate formation-related faults on production lines of an offshore oil platform. Three datasets are analyzed in this work, where each one of them is composed of a variety of sensor measurements related to the wells of a different offshore oil platform. Our methodology goes through each step of dataset cleaning, which includes: identification of numerical and categorical tags, removal of spurious values and outliers, treatment of missing data by interpolation and the identification of relevant faults and tags on the platform.

The present work designs a framework that puts together many Machine Learning classic techniques to perform the failure identification. The system is composed of three major blocks: the first block performs feature extraction: as the input data is a set of time-series signals we represent each signal using its statistical metrics computed over a sliding window; the second block maps the previous block output to a more suitable space, this transformation uses the z-score normalization and the Principal Components Analysis (PCA); the last block is the classifier, the one we adopted was the Random Forest classifier due to its simple tuning and excellent performance.

We also propose a technique to increase the reliability of the normal operation data. When handling a database composed by real data, it is usual to face a lot of mislabeled data, which can significantly jeopardize the model performance. Therefore, we deploy a technique to reduce the mislabeled samples, which presented an improvement of 7.93%, on average, reaching over 80% of accuracy in all single-class scenarios.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Machine Learning Era

Is not a novelty that applications using Machine Learning (ML) have been increasing, this area has been continuously proving its value in a variety of scenarios and under many conditions. Nowadays, applying Artificial Intelligence (AI) in a company sector is the natural course to make a business to thrive.

Even though we are far from a *Terminator* post-apocalyptic situation, we can already witness overwhelming applications such as self-driving cars [2], drones with embedded Machine Learning algorithms, and machines that outplays champions on a variety of games [3, 4]. Specific industries are embracing this revolution completely, such as health care, where professionals have proposed new techniques to provide personalized treatments to the patient [5] and advances in the image analysis area [6] are outperforming human performance in some scenarios.

This trend only became possible once we started to store and handle the data we produce every day. According to a study done by Forbes in May 2018, we produce 2.5 quintillion bytes of data daily, and 90% of the existing data were generated over the past two years [7]. These stats are impressive and the main reason behind AI achievements. Alongside the growth of ML, two related areas also came into the spotlight over the past few years: Big Data and Data Analytics, what helped to leverage ML. Roughly speaking, these areas have as their primary purpose the development of techniques to analyze a massive amount of data, extracting information that was previously hidden. They also prepare the data for the Machine Learning algorithms, helping building more robust models for the process at hand.

1.2 Oil and Gas Industry

The Oil and Gas industry followed the ML tendency, and many companies in the sector invested a lot of money and effort in this new direction. In a sector that moves billions of dollars per day, any improvement is crucial and justifies the investments to develop it. The range of applications is extensive, mainly focusing on: optimization of the rate of penetration using historical geological data, failure prediction, identification of the root cause of equipments failures, correction of drilling paths, and even analysis of supply-demand prices.

An offshore oil platform is like a living organism; it has several thousands of sensors monitoring every operation performed on it, being an open door to Big Data and ML techniques. This work focus on treating, processing and modeling those data in order to identify a failure that is relevant in terms of loss to the company. Identifying that a failure is occurring a few hours or even minutes faster has an enormous impact once the malfunctioning equipments can also be shutdown faster, what allows the maintenance team to have more time to act and, consequently, reduce the associated loss.

However, we need to limit the scope of the problem. In this dissertation, we model hydrate-related faults. The hydrate formation in injection or production lines represents a challenge to the Oil and Gas sector. When a hydrate is formed, it can considerably reduce the flow in the line, or even block it, possibly causing line jamming or malfunctioning in the involved equipment. Solving those problems is usually financially expensive and demands careful logistic to treat it. So, in face of the difficulty to deal with this phenomenon, many researchers spend their time trying to solve it or at least trying to amortize its effects through many approaches, such as to perform tests to find a better inhibitor [8], or studying hydrate kinetics to improve risk management [9]. In this work, we propose a data-driven methodology that leverages the sensors spread across the offshore oil platforms, creating a model, using the historical data from those sensors, that can identify hydrate-related faults in unseen data.

1.3 Contributions of This Work

Throughout this reading, we present a methodology that covers the entire process of handling the data from an offshore oil platform. The methodology has three major steps:

1. Identifying a critical type of hydrate-related fault in the platform, that corresponds to a considerable proportion of the production losses in that platform.

2. Cleaning the received data from relevant sensors, identifying the ones that could be useful.

3. Fitting the model by searching through a variety of hyperparameters.

We also propose a method to remove mislabeled data. This technique showed substantial improvement in almost every scenario we tested, only affecting the system during the training, and causes little impact to the model regarding its computational cost.

The results obtained in three different offshore oil platforms and three different target faults prove that the methodology is robust and ready to be deployed in a real-time situation.

1.4 Dissertation Organization

We start Chapter 2 giving a brief insight into the present relation between the Oil and Gas industry and ML. This chapter also presents the hydrate problem and justifies why it is a relevant problem to study.

Chapter 3 details the system. We present each feature extracted from the data, and discuss the normalizations we performed in the data, justifying their use and impact on the model. Next, we present the Random Forest classifier and the pre-training method, which is a method to improve the data reliability. Finally, we detail the training routine employed to fit the model and the figures of merit used to evaluate each model configuration.

Chapter 4 is the first one out of three chapters that presents the results obtained by applying the methodology described in the previous chapter. We go through the results of each processing step for the input data from a set of sensors of an offshore oil platform (Platform A). Then, we discuss the obtained results and the impact of the proposed pre-training algorithm in the performance of the best model.

Chapter 5 introduces another dataset, very similar to the previous one, but with a different predominant fault from the previous platform. Results obtained for this dataset show that our methodology works in more than a single scenario and with different sets of sensors.

Results for the third platform are presented in Chapter 6 with two distinct characteristics: this platform has a lot more occurrences and more than one relevant type of fault. So, in this chapter we discuss the results in a new fault and compare it to the results obtained on Platforms A and B. Next, we discuss the results obtained by a multiclass approach indicating that the current approach can handle more than one target fault at the same time and that the pre-training method increases the model performance once again.

Finally, Chapter 7 summarizes the conclusions of each scenario and points the reader to possible directions that this work could lead.

Chapter 2

Oil and Gas Industry Meets AI

During the last decade, many notable characters in the technology rise in the industry have been quoting Clive Humby, who stated, in 2006: "Data is the new oil" [10]. Although there are many differences between these two resources, the use of data, as the use of oil, has been powering the technology advances, such as the use of artificial intelligence in a variety of scenarios and the automation of processes in many industries.

The Oil and Gas industry impacts the economy as a whole, as it is responsible for more than half of the total world energy [11]. Even though renewable sources of power, like solar and wind, have been increasing exponentially, their impact is still far from being comparable to the one of Oil and Gas. Taking into account these facts and the expectation that the energy consumption is still going to increase, the Oil and Gas industry keeps drawing much attention.

Therefore, in the face of the Oil and Gas sector relevance, optimizing their many processes or systems has the potential to generate immeasurable gains. As stated in Chapter 1, there are many processes occurring at the same time on an offshore Oil and Gas platform. When one of those processes is unintentionally interrupted, it affects other processes around it, demanding a lot of work and time in order to make everything work properly once again. Therefore, predicting events like a non-scheduled interruption or even identifying the problem faster bring a gain to the platform production as they decrease the time and efforts to fix the problem. Thus, one way to mitigate the losses is by using the massive amount of data often produced during the operation to develop models that can generalize for the unseen data. Such approaches are known as data-driven. This new trend is not restricted to the Oil and Gas industry, it has been dragging attention of all industrial sectors [12].

In this dissertation, we propose a way to apply condition-based maintenance (CBM) to monitor processes related to a given area on the offshore oil platforms. CBM is a strategy that continuously checks the actual condition of the system and verifies if maintenance is required at the moment. In contrast to other planned and

preventive maintenance strategies, maintenance is only performed when there is a decrease in the system performance, indicating that the machinery is not working as intended. Below we list a few advantages in adopting the CBM ideology:

- It does not stop the system production to check if there is something wrong with the equipment;
- It promotes a safer environment to the workers;
- It reduces damages to other processes that are physically close to the one that presented failure; and
- It minimizes costs related to the activities scheduling, as it continuously monitors the system.

Thus, due to its appealing advantages, research in the CBM area has been increasing rapidly. According to [13], CBM has three main phases: data acquisition, data processing, and maintenance decision-making. The first phase is also the bottleneck of using CBM, as this strategy requires an existing infrastructure that allows reliable data acquisition.

The second phase of the CBM is the data processing, which receives the acquired data and processes it to best serve the next block. In this step, two areas are crucial: Big Data and Data Analytics [14]. Depending on the number of sensors and how often the data is acquired (every second, every minute or every hour, for instance), the amount of data produced can be prohibitive, requiring ways to process, clean, validate an store it in a proper manner to make it useful.

The third phase is the decision-making, and it is the step that most of the researchers devote their time to come up with innovative solutions [15, 16]. Among the lines taken by the different solutions, Machine Learning is the most explored one. Among the approaches using ML, in [17] the authors use Recurrent Neural Networks to develop an intelligent predictive decision support system (IPDSS) for CBM. The modeling of the decision-making system using ML is also addressed in [18, 19], where the authors use Support Vectors Machine [20]. In Chapter 3 we will model the decision-making step using a Random Forest classifier, alongside other ML techniques.

In the following sections, we show which faults we are addressing in this dissertation, showing their impact on the total platform loss and the challenges that these faults impose.

2.1 Hydrate-related Faults in Pipelines

In the context of this work and in the oil and gas jargon, hydrates are solid crystalline compounds formed by molecules of gas and water, under low temperatures and high pressure. The hydrate formation imposes a considerable risk to the offshore oil platform operation, when the wells and pipelines present the necessary conditions for the formation of hydrates [21].

2.1.1 Hydrates Formation Process and Consequences

The hydrate formation requires two conditions in general: high pressure and low temperature. One of the first relevant studies was made by Hammerschmidt, in 1934, in which he realized that, during winter, the natural gas pipes were clogging by hydrates, and not by ice as people believed [22]. This realization triggered the interest of people all over the world, as the industry would benefit from methods to prevent and solve hydrate-related faults [23, 24]. Figure 2.1 illustrates the temperature and pressure conditions that cause the hydrate formation. It is important to mention that this curve is highly dependent on the gas composition: as the gas density increases, the hydrate formation curve goes down, i.e., it requires a higher temperature given a fixed pressure to start the hydrate formation. In this figure, the green curve represents the hydrate formation curve for a gas which density is higher than the gas represented by the blue curve.

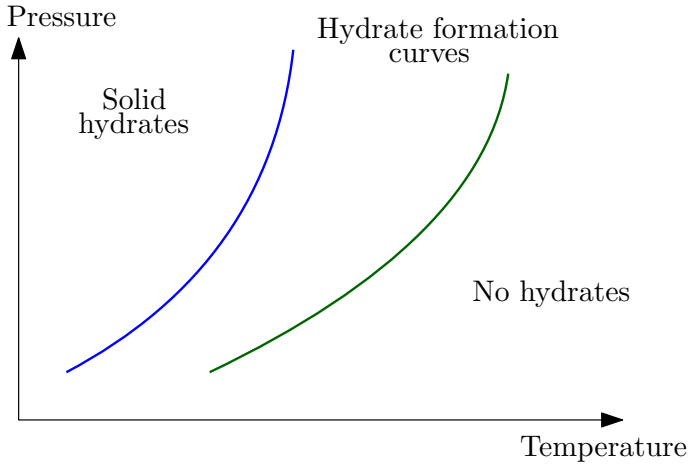


Figure 2.1: Illustration of the hydrate formation curves. In the figure, the green curve represents a gas heavier than the one represented by the blue curve. The region below the curves is free from hydrate formation, whereas the region above the curves are associated to the temperature and pressure conditions in which the hydrate formation are met.

Other aspects that can favor the hydrate formation are: agitation, or pressure pulsation; the presence of H_2S and CO_2; and the presence of free water.

The hydrate formation sometimes occurs in unexpected moments, increasing the damages in operation. One of the ways this process can be expensive for the company is when it blocks the pipelines, or reduces their flow, which in some cases can be harmful enough to stop the line production. Figure 2.2 illustrates a maintenance team removing a hydrate plug from one of the pipes, where it is clear why the hydrate removal is costly: the production of that line is completely compromised during the process.

Figure 2.2: Illustration of the removal of a hydrate plug from a pipeline showing that the process is not simple and that the hydrate plug can completely jam the pipeline. Source [1].

Other consequences vary in degrees of severity, including equipment clogging and pockets of fluid or pressure over the line. Without even considering the possibility of start a corrosion process in the pipelines, the hydrate formation is considered one of the greatest challenges of the Oil and Gas industry, which invests billions of dollars annually due to the nonexistence of a permanent solution.

2.1.2 Preventing Hydrate Formation

Due to the high cost of removing the formed hydrate plug, the best way to deal with the problem is by preventing it. There are many approaches to prevent it, and below we discuss a few of them.

1. Dehydration: Removing water reduces the dew point of the fluid and consequently prevents it from reaching the temperature and pressure conditions to the hydrate formation. One way to perform the dehydration is by injecting glycol, usually triethylene glycol (TEG) [27].

2. Kinetic rate inhibitor: Due to the high cost of glycol injection, kinetic inhibitors are a cheaper solution that slows down the hydrate formation [25, 26].

3. Temperature control: In some systems, it is possible to control the temperature along the pipeline directly. So, it is possible to maintain the temperature above the hydrate formation curve, despite its pressure.

In the next chapter we will present our approach to address the hydrate formation prevention in detail. Using a data-driven approach, we rely on very few specific knowledge domain and take advantage of the pre-existent structure on the platform: thousands of sensors are already installed.

2.1.3 Hydrate Failures Relevance on an Offshore Oil Platform

To verify the impact of the hydrate formation in the analyzed offshore oil platforms, we consider its total impact in terms of loss and number of occurrences. All failure events of a group of ten offshore oil platforms from January 2010 to June 2017 are registered in a document via a platform we called Production Loss Database (PLD). In this report, each event is identified using along three different levels of details: Reason Group, Group, and System.

In Figure 2.3 there is a representation of one of the Reason Groups ("Gathering") and its ramifications. We are representing the "Gathering" Reason Group, as it is the one that branches into hydrate related events, specified under System classification.

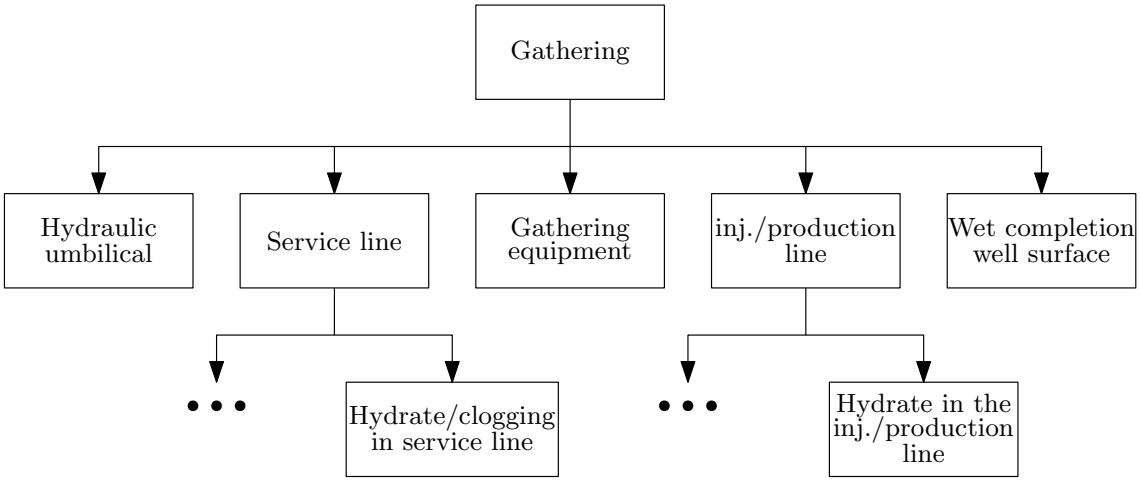


Figure 2.3: Illustration of one of the Reason Groups and its ramifications, according to PLD. The first row (from top to bottom in the figure) represents the Reason Group "Gathering", which includes all hydrates-related failures. The second row represents the five Groups that lie under Gathering, and we branched two of them to show the Systems related to hydrate formation in the third row.

Our analysis begins by verifying the Reason Group "Gathering" relevance when compared to the other possible Reason Groups. In Figure 2.4, one can notice that the "Gathering" importance represents 22% of the total number of failure occurrences and 27% of the total production loss, considering all ten platforms. Among all Reason Groups, "Gathering" is the second most significant according to both criteria.

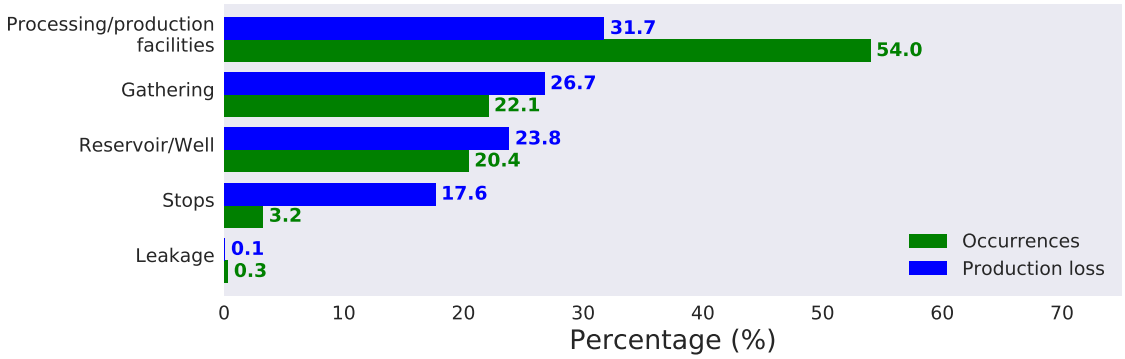


Figure 2.4: Breakdown of all events occurrences and cumulative production loss according to the Reason Group. The "Processing/production facilities" is the most common Reason Group, but "Gathering" has its importance and represents over a quarter of total production loss.

In Figure 2.5 we have a similar analysis, but now we are restricted to the events under the Reason Group "Gathering". In this case, one can claim that the Group "Injection/production line" represents the majority of problems in this group of platforms. So, at this point we could make clear decisions to narrow our target fault search without losing the sense of impact the fault represents.

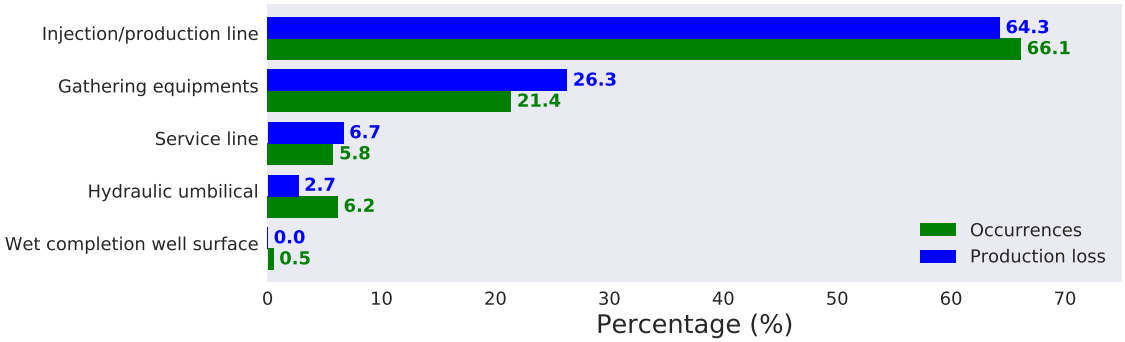


Figure 2.5: Breakdown of the "Gathering" events occurrences and cumulated production loss according to the Reason. It is possible to notice the dominance of the Reason "Injection/production line", showing the importance of its study.

Finally, Figure 2.6 illustrates the contribution of all events from the "Gathering" Reason Group according to its System. From this figure, the hydrate-related failures are the most common and represents the most relevant impact in the platforms considered in this work.

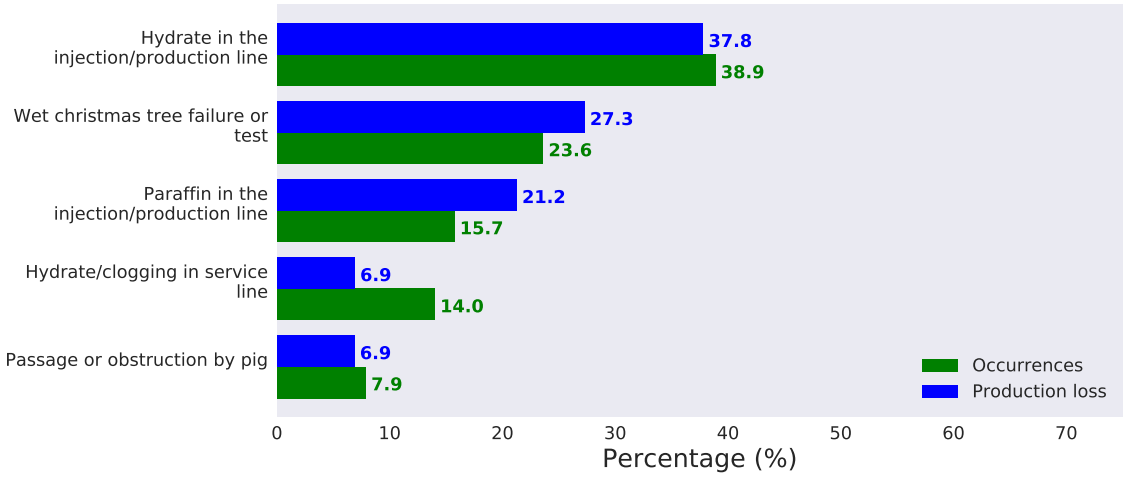


Figure 2.6: Breakdown of the "Gathering" events occurrences and cumulated production loss according to the System. According to this classification, there are many possible target faults that have enough number of occurrences and presents a considerable percentage of the production loss.

The methodology we are going to propose further in this text is an oil platform-basis approach, in which we apply every step in each database at a time. This distinction is necessary due differences of each offshore oil platform: not only the dominant hydrate related failure can change, but also which tags are available for each one of them. To avoid restricting our search for only one or two types of fault, we chose seven possible hydrate related faults to explore:

1. Hydrate in the injection/production line
2. Hydrate/clogging in gathering equipments
3. Hydrate/clogging in service line
4. Inorganic incrustation in injection/production column
5. Inorganic incrustation in injection/production line
6. Organic incrustation in injection/production line
7. Paraffin in the injection/production line

2.2 Conclusions

In this chapter we discussed the rising of ML applications in the Oil and Gas industry. We further analyzed a specific type of faults that is very common in offshore oil platforms: hydrate-related faults. These failures represent a major challenge to the oil and gas sector and have been dragging the attention of investors over the past years, and due to its complexity there is not a standard solution to prevent the occurrence of these faults, specially when applied to offshore oil platforms injection and production lines.

Thus, we proposed a data-driven approach, which will be explained in details in the next chapter, that takes advantage of the thousands of sensors installed across the oil platforms. This approach performs the identification of hydrate-related faults using the data collected from three offshore oil platforms in the period of January 2010 to June 2017. We also showed that during this period, the hydrate-related faults had relevant impact on the production process of those platforms regarding total loss and number of occurrences.

Chapter 3

System Overview

This chapter aims to explain every detail of the deployed classification system, from how the data is first handled to what the system will give as output. Figure 3.1 shows a block diagram of the system, which is composed by 3 major parts: first of all, the input feeds a block that will extract features from the raw data; next, these features are transformed, in a manner to keep only meaningful data; and then, the processed data is modeled, according to a given classification task.

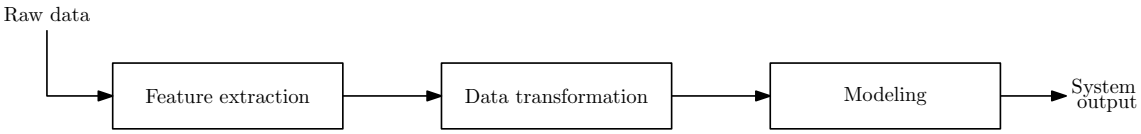


Figure 3.1: Overview of the system used in this work. The block "Feature extraction" treats the raw data and extracts some features from it; the "Data transformation" block maps the data from the feature space to a suitable space, and the block "Modeling" classifies the data by associating a label to it.

In Section 3.1, we present the details of the first block of the system, explaining which features are extracted from the data and how it is done. Section 3.2 explains the second block of the system, giving the details about which transformations are applied and why we chose them. Section 3.3 explains the Random Forest classifier, whereas Section 3.4 explains a novel technique that is applied to increase the reliability of the normal data. Section 3.5 goes through some details about the training routine applied, Section 3.6 discusses the figures of merit used to evaluate the results, and, finally, Section 3.7 shows the flow of normal samples throughout the system.

3.1 Feature Extraction

The first block of the system receives the cleaned data from the selected tags. The data at its input is a time-series, sampled minute by minute, expressed as $\mathbf{X} \in \mathbb{R}^{N \times n}$, where n is the number of selected tags and N is the number of samples of each tag.

Prior to this stage, the data has only been cleaned, and no further transformation has been applied to it.

At first, one can expect that the raw data contains all the information, and it is in its best form. Although the first proposition is correct, the second one is not. And it is here where the "Feature Engineering" intervenes, trying to find a better space to represent the data. This process can be done by infusing domain knowledge about the process at hand or just by expressing the data via its many statistical features.

In this work, we adopted a data-driven approach, computing from each tag, nine statistical features. Since each tag is a single time-series covering all period, these features are calculated considering a sliding window of length N, considering the actual sample and the $(N-1)$ previous ones:

- Mean: Computes the arithmetic mean along the sliding window

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i, \tag{3.1}$$

  where x_i is one of the window's samples.

- Standard deviation: Computes the standard deviation along the sliding window. This measure is used to access the spread of the data around its mean:

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu)^2}. \tag{3.2}$$

- 5-numbers summary: This set of features represents the minimum, maximum, median, and first and third quartiles of the sliding window. While the median measures the center of the distribution, the minimum and maximum deliver its range. The first and third quartiles give the idea of how the distribution is spread, especially when compared with its minimum and maximum values. These metrics often provide enough detail about the dataset to satisfactorily solve the classification problem by themselves [28].

- Skewness: It is the third standardized moment of a random variable, and is defined as:

$$h = \frac{\mathrm{E}[(X-\mu)^3]}{(\mathrm{E}[(X-\mu)^2])^{\frac{3}{2}}}. \tag{3.3}$$

  This number measures the asymmetry of the distribution [29]. If $h = 0$ then the data is symmetric, and then its mean is equal to its median.

- Kurtosis: It is the fourth standardized moment, and is computed trough the following formula [30][1]:

$$k = \frac{\mathrm{E}[(X-\mu)^4]}{(\mathrm{E}[(X-\mu)^2])^2} - 3. \tag{3.4}$$

  This metric is used to evaluate the tails of each distribution. If $k \ll 0$ there are no tails, whereas if $k > 0$, the distribution presents tails heavier than the ones in the normal distribution.

As aforementioned, these features are evaluated over a sliding window, transforming a set of $N \times n$ samples (with N time samples from n tags) into a vector of nf elements (in the case where f features are calculated per tag). Figure 3.2 shows how the transformation occurs, and it can be interpreted as a function that maps a real matrix $a \times b$ into a real matrix $c \times bf$, i.e., $f : \mathbb{R}^{a \times b} \longmapsto \mathbb{R}^{c \times bf}$. Considering a sliding window of length N and step s between two consecutive windows, the function f yields an output with $c = \lfloor \frac{a-N}{s} \rfloor + 1$ instances. Thus, $\mathbf{X}_{\text{tr}} = f(\mathbf{X})$ yields a $\lfloor \frac{M-N}{s} \rfloor + 1 \times nf$ matrix, considering M time instants represented in the input matrix.

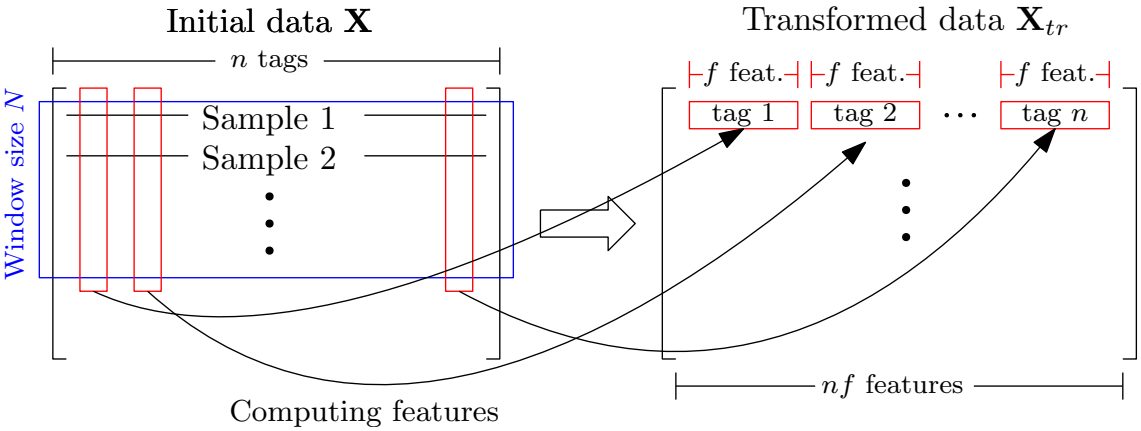


Figure 3.2: The matrix on the left side represents the initial data $\mathbf{X}$, which has n tags. A a transformation is performed on a sliding window covering N rows will slide along all its columns with a step of s rows between two transformations. When transforming the samples inside a window, for each tag, k features will be computed. So, the transformed data has nk columns.

It is worth noticing that this process adds two hyperparameters to our model: the window size N and the step s. Considering that $M \gg N$, N will have almost no influence on the number of samples of the transformed matrix. On the other hand, the step s drastically reduces the amount of data after extracting the features, hedging its range when looking for the better set of hyperparameters.

[1] the -3 factor in this equation guarantees that the kurtosis of the normal distribution is set to $k = 0$.

Since the data have been transformed, the corresponding labels have to be modified as well. An instance of the set of features was computed considering N samples from the original data, each of which had a label assigned to it. Thereby, the label associated to a given window is computed as the most common among the N labels inside it.

3.2 Data Transformation

The second block of the system is responsible for representing the data in a more suitable space by means of a transformation. In this work, this block consists of two steps: the first one is the data normalization, and then a principal component analysis (PCA) will be applied.

3.2.1 Data Normalization

Many techniques are available to the user when assembling a machine learning model. Deciding which one to use is a matter of experience, preference and how many computational resources are available. This step has the simple purpose to re-scale all features to the same range, which can be crucial depending on what methods will be used further in the system. Having in mind that the previously calculated features lie on different ranges, normalizing the data is a good practice, and the way we have opted to do that in this work is using the z-score normalization.

This way to normalize the data is also known as standard normalization, and is achieved by the following equation:

$$Z = \frac{X - \mathrm{E}[X]}{\sigma_X}, \tag{3.5}$$

where X is a random variable and σ_X its standard deviation.

Chapter 2 presented an insight on how the data is collected, and in the further chapters the set of tags of each platform will be detailed. However, it can be anticipated that the tags come in a variety of units, which is a significant reason to employ z-score over others normalization methods, like log-transformation [31].

According to the block diagram presented in Figure 3.1, the "Feature extraction" block precedes the "Data transformation" block. So, the normalization is performed in the feature space, and each one of those features are normalized separately, considering its whole duration. It is also worth mentioning that the z-score is part of the model, and then, performs the normalization on the training set saving the median and variance of it. When applying the model to predict (validation or test sets), the normalization is made via these stored values.

3.2.2 Principal Component Analysis

After the feature extraction block, since 9 features are computed for each of the n tags, each sample has $9n$ features. One can claim that as the number of features increases, the information about the data also increases and then, discerning which class a sample belongs will be easier. This reasoning, however, is not entirely correct; projecting a model that properly handle high dimensional data demands more observations, and will probably be more complex [32]. This dilemma, known as the "curse of dimensionality", is vastly discussed in the literature and is associated to the problem of having an unnecessarily large number of features, however counter-intuitive it may be [33].

The PCA brings a way to deal with this problem, by reducing the dimensionality of the feature space while keeping its essential information. Even though this is the most appealing use of the PCA, sometimes just transforming the data is enough to boost the performance of a system. This improvement comes from the fact that the PCA projects the data on an orthogonal space in a manner to maximize the variance of the projected data. Formalizing this statement: PCA starts searching for a direction $\mathbf{c}_1$, with unit length, such that the projection of the centered data $\mathbf{X}$ has maximum variance, which can be expressed as:

$$\sigma_1^2 = \frac{1}{n-1}\mathbf{x}_{proj_1}^T\mathbf{x}_{proj_1} = \frac{1}{n-1}\mathbf{c}_1^T\mathbf{X}^T\mathbf{X}\mathbf{c}_1 = \mathbf{c}_1^T\mathbf{C}\mathbf{c}_1, \tag{3.6}$$

where $\mathbf{C} = \frac{1}{1-n}\mathbf{X}^T\mathbf{X}$, and $\|\mathbf{c}_1\|_2 = 1$.

Using Lagrangian multiplier one wants to maximize the cost function below:

$$L = \mathbf{c}_1^T\mathbf{C}\mathbf{c}_1 - \lambda(\|\mathbf{c}_1\|_2 - 1). \tag{3.7}$$

Differentiating it with respect to $\mathbf{c}_1$:

$$\mathbf{C}\mathbf{c}_1 - \lambda\mathbf{c}_1 = 0 \Rightarrow \mathbf{C}\mathbf{c}_1 = \lambda\mathbf{c}_1. \tag{3.8}$$

So, the first principal component, $\mathbf{c}_1$ is an eigenvector of the data covariance matrix. Substituting the solution found in Equation (3.8) in Equation (3.6):

$$\sigma_{max}^2 = \mathbf{c}_1^T\mathbf{C}\mathbf{c}_1 = \mathbf{c}_1^T\lambda\mathbf{c}_1 = \lambda. \tag{3.9}$$

Thus, the first principal component is the eigenvector associated to the maximum eigenvalue of $\mathbf{C}$. Calculating the next component can be done following the same steps, but now considering a new constrain: $\mathbf{c}_2^T\mathbf{c}_1 = 0$, once the directions are orthogonal.

Another result obtained by applying PCA to the data, is a geometric perspective: PCA minimizes the representation error when only a fraction of the principal

components are used. Once the data is projected on the first principal component, recovering the original data from this projection yields the following error[2]:

$$\begin{aligned} e_r &= \|\mathbf{X} - \mathbf{X}\mathbf{c}_1\mathbf{c}_1^T\|^2 \\ &= \text{tr}((\mathbf{X} - \mathbf{X}\mathbf{c}_1\mathbf{c}_1^T)(\mathbf{X} - \mathbf{X}\mathbf{c}_1\mathbf{c}_1^T)^T) \\ &= \text{tr}(\mathbf{X}\mathbf{X}^T - \mathbf{X}\mathbf{c}_1\mathbf{c}_1^T\mathbf{X}^T - \mathbf{X}\mathbf{c}_1\mathbf{c}_1^T\mathbf{X}^T + \mathbf{X}\mathbf{c}_1\mathbf{c}_1^T\mathbf{c}_1\mathbf{c}_1^T\mathbf{X}^T) \\ &= \text{tr}(\mathbf{X}\mathbf{X}^T) - \text{tr}(\mathbf{X}\mathbf{c}_1\mathbf{c}_1^T\mathbf{X}^T) \\ &= \text{tr}(\mathbf{X}\mathbf{X}^T) - \text{tr}(\mathbf{c}_1^T\mathbf{X}^T\mathbf{X}\mathbf{c}_1) \\ &= \text{tr}(\mathbf{X}\mathbf{X}^T) - (n-1)\mathbf{c}_1^T\mathbf{C}\mathbf{c}_1 \end{aligned} \tag{3.10}$$

So, minimizing e_r is equivalent to maximizing $\mathbf{c}_1^T\mathbf{C}\mathbf{c}_1$, which is the variance of the projection, according to Equation (3.6), and is obtained using the first principal component as the projection direction.

The result of applying this transformation is a matrix with the same dimensions of the input data $\mathbf{X}$, but with its first column having maximum variance. If we perform this recursively for the other principal components in decreasing order of eigenvalue magnitude, the same stands for the other columns of the transformed data: the variance of the second column is greater than the other variances and so on. Variance of a feature is a measure of how spread is this feature, and it is connected with how much information it carries. For example, suppose that a feature has variance equal to 0, i.e., it is constant. Thus, using it further in the system brings no advantages, just increasing the computational cost.

After applying PCA on an $n \times m$ matrix, the output of this transformation is another $n \times m$ matrix, but now, has uncorrelated columns and they are sorted according to their variance. Using PCA as feature selection exploits this ordering by defining as the explained variance of a component the ratio between its variance and the sum of all components variance. So, one can pick a certain number of components that "explains a determined amount of the signal's variance".

Figure 3.3 brings an illustration of the sum of this explained variances, where the original data has 63 features. One can notice that the 33 first components (the 33 with the highest variance) explain 95% of the overall variance, and 41 components cover 99% of it, which are much less than the 63 original ones. So, one can reduce the number of components considering how much information can be put away and how much resources are available.

[2]The use of $\|.\|$ is extended to matrices, where $\|\mathbf{A}\|^2$ is the sum of the squares of all elements of $\mathbf{A}$.

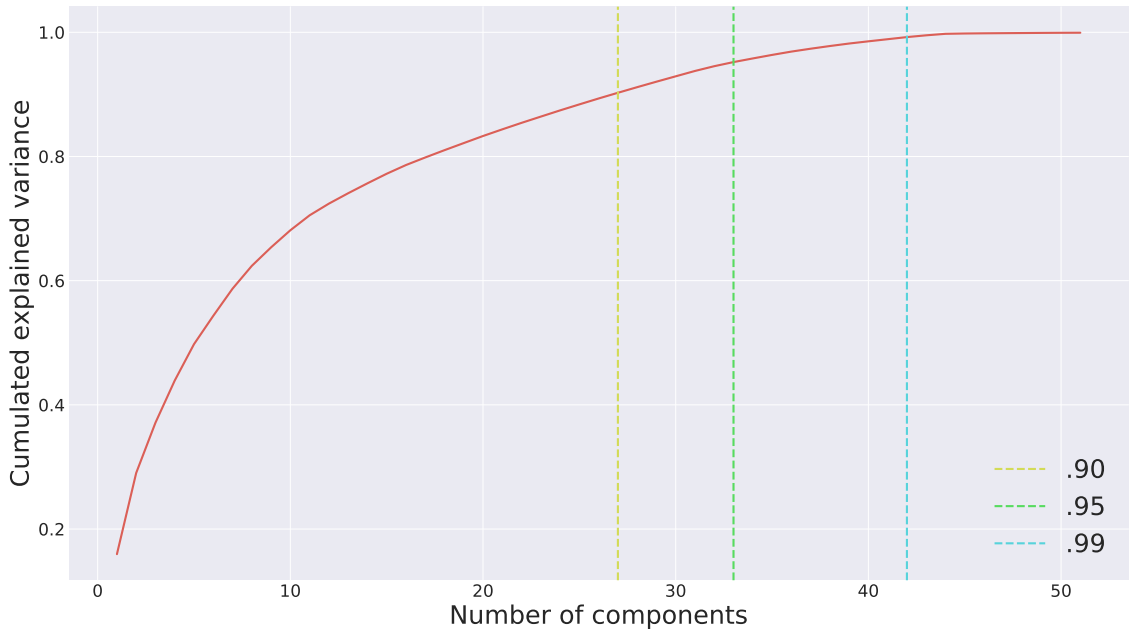


Figure 3.3: The cumulated sum of the variances from the components after the use of the PCA. Starting with the component with the highest variance, one can notice that the sum converges to 0.99 with much fewer features than the original data (in this case, 63 features).

3.3 Random Forest Classifier

In the proposed system, this last block associates a class label to the input data. Since the datasets used in this work have labels, the classification method will be supervised. One of the further necessities of the system is a probabilist output, i.e., a number that carries the certainty of the classification. In this work, we restrict ourself to analyzing only one algorithm: Random Forest, which also allows the system to solve a multiclass problem.

The Random Forest classifiers [34] have been used in a wide variety of scenarios: gene selection [35], remote sensing [36], and prediction of protein behavior [37], among others. They consist in ensembles of decision trees, called weak learners in this context, that altogether create a stronger learner. Each decision tree is trained based only on a restricted subset of the data and only on a subset of the features. In the Random Forest classifier, each tree outputs a class as its prediction, and by counting all "votes", the ensemble of these trees considers the most common vote as the model response, as illustrated in Figure 3.4.

Since each tree is trained by a random subset of the data, it tends to be uncorrelated from other trees. At this point, the algorithm is known as tree bagging [38] and differs from the Random Forest by only one aspect. Suppose that a particular feature f is good on the task of classifying a given sample. So, every tree that has access to f will use it during its training, and then, those trees will be correlated

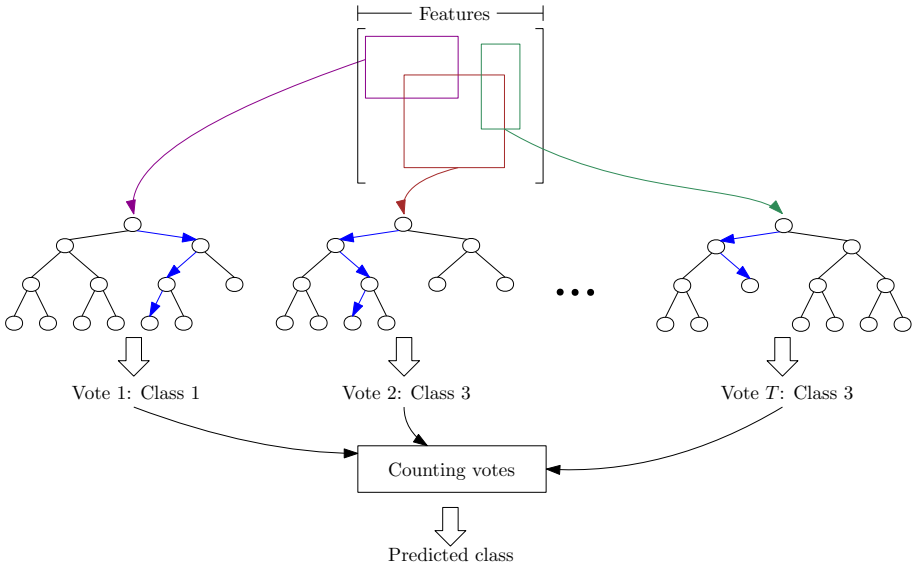


Figure 3.4: Each tree receives a random subset of the data (colored boxes on the top of the figure) and then, after training, outputs a class. After evaluation, all votes are counted, and then the predicted class is chosen by the majority of votes.

with each other. To handle this, each time a tree considers a split, it will decide considering only a small random subset of the used features.

Although in the original Random Forest paper the author stated that it does not overfit, in [39] the authors showed that it could overfit for noisy data, which is the scenario for most real applications. So, search for hyperparameters that avoid overfitting is mandatory in this work.

Another key point of using the Random Forest is the fact that each tree is trained separately from each other, so, parallelizing the training is an easy task. All the system have been developed using Python 3, and in the case of Random Forest, there is a great implementation of it on the scikit-learn [40], based on [34].

In this work, two Random Forest hyperparameters will be explored:

- Max tree depth: This parameter restrains how deep each tree can go. Letting it loose, i.e., not controlling it, usually leads to overfitting, since the training accuracy will be close to 1.

- Number of estimators: As each estimator uses only some training samples, a reasonable amount of trees is needed to train with all samples. However, an unnecessary increase in this parameter sometimes only increases the complexity of the model, delivering no increase in the performance [41].

3.4 Pre-training

Chapter 2 presented the dataset and the cleaning process initially applied to prepare the data. When dealing with real data, extra measures are needed to increase the reliability of the data. In this work, the labels are created based on annotations made by the operators on the platform at the moment the fault occurred; so, it is expected that it contains some errors. However, the presence of those mislabeled samples usually decreases the predictive power of the model [42].

Another aspect present on these datasets is the abundance of data corresponding to the normal class. This galore of normal data gives the opportunity to perform some data selection on the normal samples to only forward reliable samples to the training phase. So, in this section we consider a stage incorporated to the classifier training that selects reliable normal samples, that will be referred to as the pre-training phase.

The data will initially feed the pre-training block, and there, a $k-$fold cross-validation[3] will be performed. During cross-validation, each sample appears $(k-1)$ times in the training set. So, each time it appears, if the prediction of this sample during the training is equal to its true label, a score associated to this sample will be incremented by one unit. Each sample has a score equal to zero at the beginning of this phase and can reach up to $(k-1)$ in the end. After the cross-validation, if a sample has a score greater than a given reliability threshold τ, it will be sent to the training phase, otherwise, it will be removed. The hyperparameter τ controls the selectiveness of the pre-training algorithm, as it imposes how many times a sample needs to be correctly classified. Figure 3.5 brings the flowchart of this algorithm.

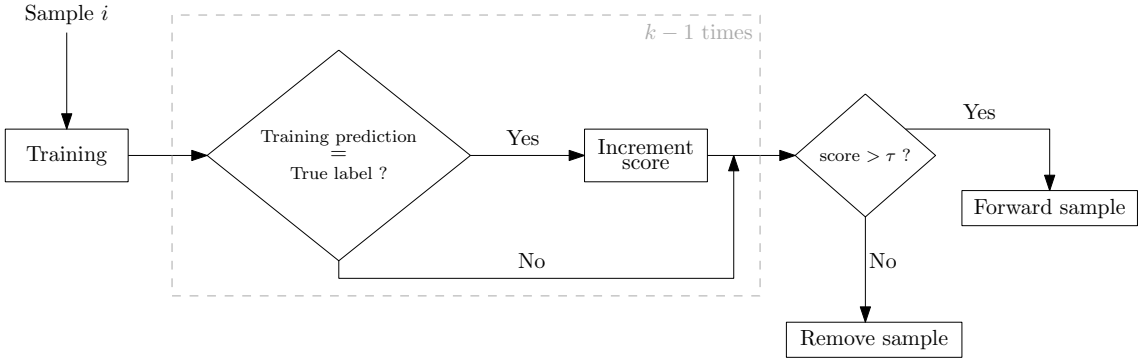


Figure 3.5: Each sample i joins the training set $k-1$ times during cross-validation. After each training, the algorithm compares the predicted value during training with its true label; if they are equal, it increments the sample score by one unit. When the training ends, the score is compared with a given threshold, and in the case that the score is greater, the respective sample is forwarded to the next block.

[3]Although this term is being used in this section, next section will bring more details and specifications on this.

It is expected that increasing the reliability threshold τ, the algorithm becomes more selective, and, consequently, less samples will be forwarded to the next phase. Since the dataset used in this work has a limited number of samples from the target class, this step will be applied only on the normal samples, which are abundant.

This method requires extra caution when choosing the training algorithm. One can notice that, if the model overfits the data during training, no samples will be removed and the method will be pointless. So, it is necessary to use a method with high accuracy that will not overfit. In this work we used the Random Forest classifier, limiting the maximum depth of each tree by 3 and the number of estimators to 100.

In this work, many values for τ will be tested (notice that $0 \leq \tau \leq k-1$), and the algorithm will be performed recursively many times, i.e., the i-th round feeds the $(i+1)$-th round and so on.

As a measure of how selective the final model will be and to avoid overfitting, an extra validation set will be used. This set will not be cleaned by the pre-training phase and it will be used to decide between different sets of the pre-training hyperparameter (τ and number of rounds). Considering that this set will not be cleaned, it probably contains some mislabeled samples. Note that, it is expected that the accuracy on this samples decreases, as the classifier has not been trained on unreliable samples.

The accuracy on the extra validation set needs to be analyzed in order to establish a trade-off between accuracy gain and the chance to overfit the model. This analysis is qualitative, based on how much data contamination is expected. In Subsections 4.4.2, 5.4.2, and 6.4.2 we explain the choice of pre-training hyperparameters in each scenario. Figure 3.6 depicts the configuration of the system with the addition of the pre-training block preceding the modeling block so that mislabeled samples can be removed.

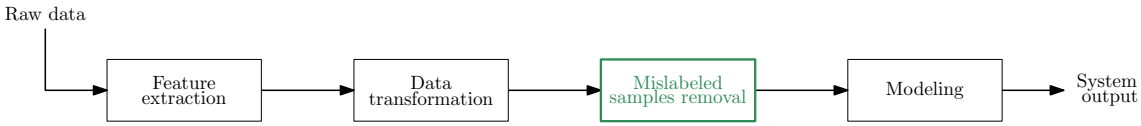


Figure 3.6: Each sample i joins the training set $k-1$ times during cross-validation. After each training, the algorithm compares the predicted value during training with its true label; if they are equal, it increments the sample score by one unit. When the training ends, if the score is larger than a given threshold, the respective sample goes to the next block.

3.5 Training Routine

This section explains two details about the training phase. All datasets used are very much unbalanced, with the number of target samples being close to only 0.5%

of the number of samples from the normal class. So, to treat the problem as a balanced classes problem, the normal class will be subsampled so that it will have only R times more samples than the target class. The R hyperparameter will be varied during the training to study its effect on the predictions.

This step is a good practice in the context of this work since correctly identifying the target class is more important than a few false positives. Without the class rebalancing, the model achieves better overall accuracy results, but the target class accuracy decreases drastically, which is not convenient to the application at hand.

At this point, the model has already a considerable number of hyperparameters (maximum tree depth, number of estimators, rebalance ratio, window size, window step). It is necessary to go through all the combinations of these hyperparameters to decide which set delivers the best model. The next subsection presents how this is done in this work.

3.5.1 Cross-validation

When training a model, a good practice consists on splitting the data into three groups: training, validation and test set. The primary goal of the validation set is to measure the performance of the trained model, and, depending on its performance, modifications will be made to the model, and it will be trained again. This process usually has to be carried out several times until the model reaches an acceptable performance. It is worth noticing that the test set can not be used in this loop. This is so because this set must be kept untouched since it is the one used to assess the final performance of the model, or its generalization capability [43].

In theory, this setup is ideal, but typically there is not enough data available for that, and cross-validation is a way to work around this problem. So, this method is a way to do *model selection*, and how to use it varies according to the data. Two strategies will be used, one for the target class samples and another one for the normal samples.

Normal Samples

Probably, the most widespread way to use the cross-validation is the k-fold strategy. In this approach, the training set is split into k subsets (each subset is called a fold). Figure 3.7 brings an illustration of this process, whereupon k iterations are performed, following two steps in the iteration i [44]:

1. A model is trained on all folds, with the exception of the i-th fold;

2. The trained model is evaluated on the i-th fold.

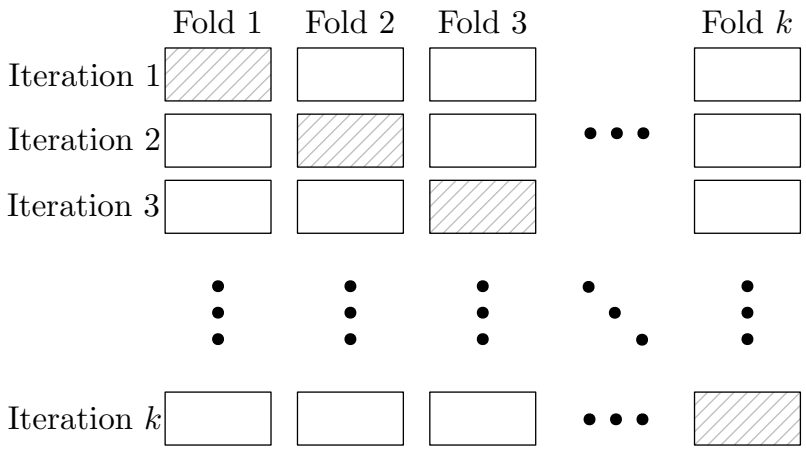


Figure 3.7: Illustration of k-fold. Each iteration uses the hatched fold as validation set and the white folds as the training set. One can notice that each fold is used as the validation set only once.

As the loop ends, the result of this set of parameters is the mean of the validation results over all iterations. From the per fold result it is also possible to compute the variance of the accuracy, enriching the result with the idea of how stable is the process, or, in other words, how robust is the model.

Applying this approach requires to train the model k times for each combination of hyperparameters explored, making it cumbersome, computationally speaking, especially when using a complex algorithm to train the model.

However, the data consists of time-series signals, and there is an alternative when doing the k-fold cross-validation. Although the traditional k-fold can perform well when dealing with time-series, the intrinsic time correlation of the data appeals for an alternative [45]. This alternative will be referred to as time-series k-fold and is a variation of the standard k-fold. For this variation, in the i-th iteration, instead of training with the remaining $k-1$ folds, the training set consists of the first i folds and the validation is the next fold, as illustrated in Figure 3.8.

Target Class Samples

The target class in this database is a given fault during operation and is represented by periods of time along the data duration. So, all the labels representing this class come as blocks, covering all the corresponding fault period, Figure 3.9 illustrates this.

Each time a failure occurs, it creates an event and all labels along its duration are associated to the code assigned to the failure. In this variation of cross-validation, samples from the same event will never be in separated sets, i.e., they will be either part of the training set altogether, or compose the validation set. This concept is adopted since otherwise a sample of the training set could carry very similar

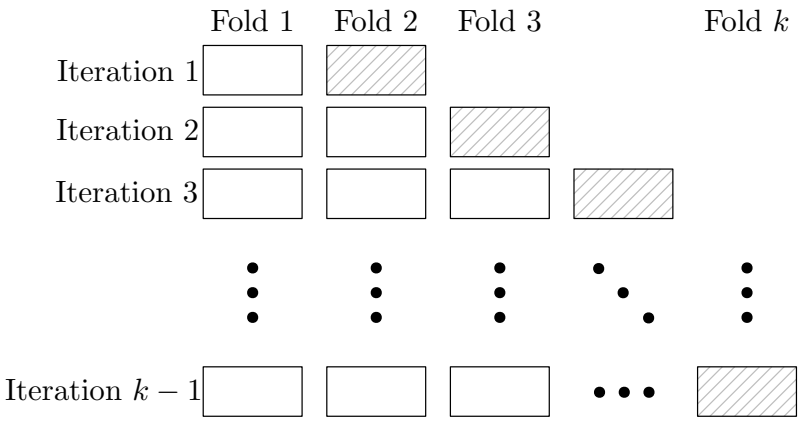


Figure 3.8: Illustration of time-series k-fold. The all time-series is split into k folds, and at the i-th iteration, the first i folds are used as training set, and the validation set is the $(i+1)$-th fold.

information to the one carried by samples of the validation set, and this would contaminate the training process, jeopardizing the ability of the validation set to evaluate the generalization capability of the classifier.

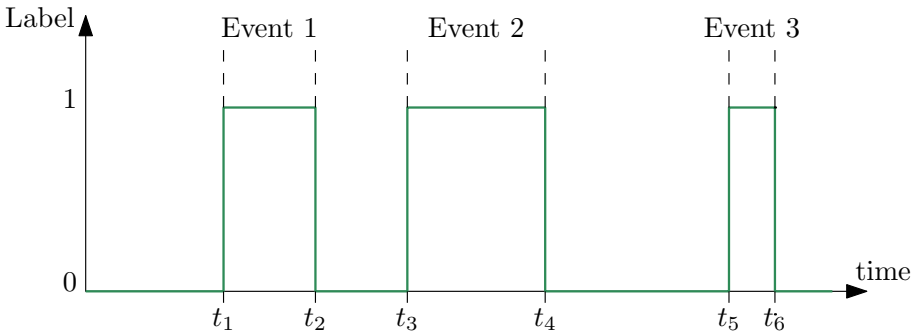


Figure 3.9: The green curve represents the label over the time, and it is equal to 1 during faults periods. In this example three events are occurring.For example, all samples between timestamps t_1 and t_2 belong to Event 1. According to this cross-validation variation, all samples from an event should stay together in a single set.

3.6 Figures of Merit

To evaluate the models, some figures of merit will be used. Accuracy is the most common metric used to assess model performance and represents the fraction of predictions that were correctly classified. Mathematically, it can be expressed as:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}, \tag{3.11}$$

where TP stands for *True Positive*, FP means *False Positive*, TN means *True Negative*, and finally, FN stands for *False Negative*.

Analyzing the accuracy by itself could mislead the user to poor decisions. In a variety of scenarios, correctly detecting a fault is much more important than precisely classifying a normal sample. In other words, TN and TP do not have the same value in the given scenario. So, to support the decisions between the models, other figures of merit will be analyzed.

Precision represents the ratio between the correctly classified positive samples to the total positive predicted samples, i.e.:

$$\text{Precision} = \frac{TP}{TP + FP}. \tag{3.12}$$

In the case we have Precision equals to 1, this means that every time the classifier predicted a sample as positive, it was indeed from the positive class. But, this measure does not reflect the FN, i.e., the positive samples that the model classified as negatives.

Recall is used alongside precision to address this problem. It represents the fraction of the positive samples that were correctly classified.

$$\text{Recall} = \frac{TP}{TP + FN}. \tag{3.13}$$

If Recall equals to 0.8, it means that 80% of the target class samples were correctly classified. One can notice that this measure does not take into account the number of normal samples misclassified as targets, but precision covers this.

The F_1 score is the harmonic mean between precision(P) and recall (R):

$$\text{F}_1 = \frac{1}{\frac{\frac{1}{P} + \frac{1}{R}}{2}} = \frac{2P.R}{P + R}. \tag{3.14}$$

The harmonic mean is taken once precision and recall are ratios, and the harmonic mean is the usual way to combine ratios with equal weight. The F_1 is a particular case of the F-measure, where both precision and recall are intended to have the same value [46].

Another figure of merit used to evaluate the results is the visual result on each event. Suppose a scenario that the model correctly classifies the beginning of the event but misses the entire last third of the event period. In a second scenario the classifier correctly classified 80% of the fault, but in a erratic manner, as illustrated in Figure 3.10. Even though the second scenario performs better under the previously discussed figures of merit, on real applications, the first model is much more reliable. Correctly detecting the start of the event enables the operator of the system to deal with the failure appropriately. So, correctly identifying the beginning of the fault has more value than identifying its end.

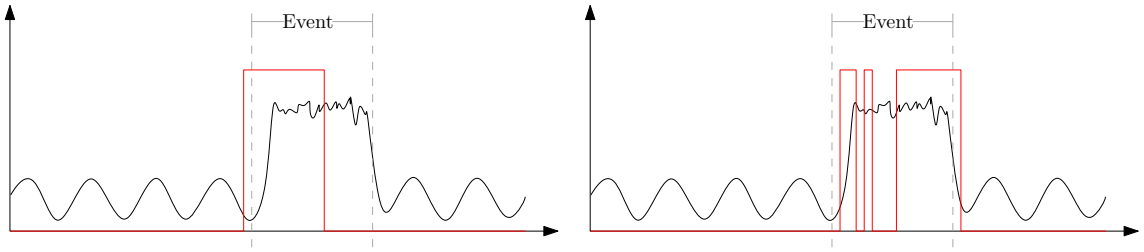


Figure 3.10: The left graphic illustrates a first scenario, where the model correctly identifies the start of the failure. The right graphic represents a second scenario which has better performance than the first model in terms of precision, recall and F_1, but is less useful in practice since it has problems identifying the beginning of the fault.

3.7 Tracking Normal Samples

Throughout this chapter the proposed system was presented by explaining each of its blocks with enough details so as to allows its replication by a careful reader. The normal samples have a long trajectory from the moment when they enter the system to the prediction phase. To clarify their journey, this section encapsulates the steps taken by them.

First, the normal data are sampled in order to balance the classes. Considering that normal samples are much more present than the ones from the target class, this balancing is performed by sampling the normal data. Past this step, the sampled normal data is split into two sets: the extra validation set, which contains the same number of samples as each validation fold of the model training, with the rest feeding the pre-training block.

As discussed in Section 3.4, the pre-training block has the purpose of purging unreliable samples from the normal dataset. So, to guarantee that this block outputs the necessary number of samples to the next block, the following equation approximates the number of samples required on the pre-training:

$$|\mathcal{T}_{\text{pre}}| = \frac{1}{(1-\delta)^{\text{rnds}}} R|\mathcal{H}|, \tag{3.15}$$

where *rnds* is the number of rounds of the pre-training, R is the balance ratio (see Section 3.5), $|\mathcal{H}|$ is the number of samples of the target fault, and δ is a scalar that regulates how much it is needed to compensate the normal samples removal each round of the pre-training. In this work we empirically concluded that using $\delta = 0.15$ brings a good estimate of the number of inputs to the pre-training step for one to have enough samples on its output, but it is not precise. Thus, some adjustments may be necessary in order to make available precisely $|\mathcal{T}| = R|\mathcal{H}|$ samples on the training. The whole process is illustrated in Figure 3.11.

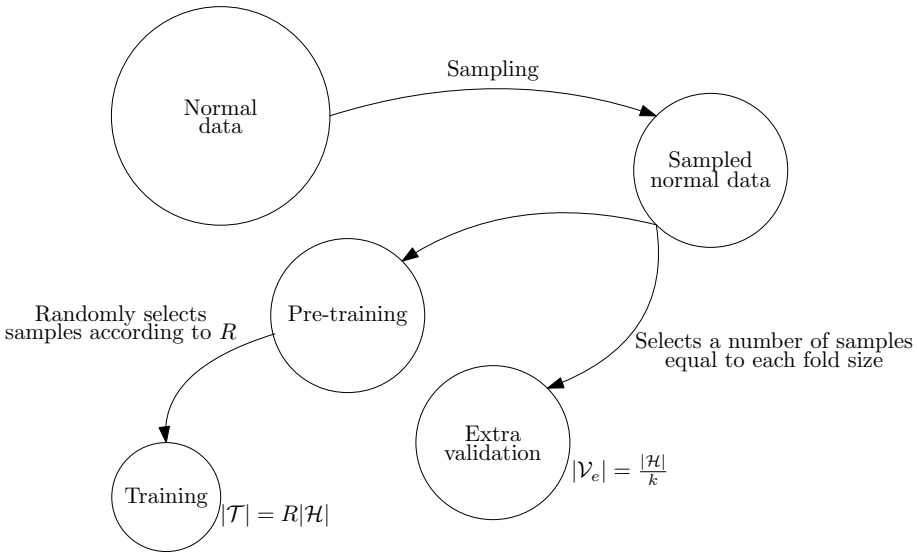


Figure 3.11: Path of the normal data samples. At the start, they are sampled so that there are enough data to support the rest of the model. Then, the data follows to two separated paths: they can feed the extra validation set or go to the pre-training phase. The pre-training will purge some samples, creating the need for more samples than it is necessary on its output. After pre-training, the data is sampled one more time, to finally meet the desired condition of having $R|\mathcal{H}|$ samples.

3.8 Conclusions

Throughout this chapter we presented the system in detail, following the data from the first block of the system all the way to its output. Since the data used in this work belongs to an oil company, the code to reproduce the system is not available for confidentiality reasons, but the present chapter was written so that a careful reader, should be able to reproduce the model described here utilizing Python 3.6 and scikit-learn.

The proposed system follows all the standard steps when first handling a database. In addition to that, we proposed a way to increase the reliability of the normal samples and explained how to evaluate its performance. Each of the Chapters 4, 5, and 6 of this text will present further details about the data cleaning steps and the way we restricted our problem in order to deal with data coming from three different offshore oil platforms, along with their respective results.

Chapter 4

Results for Offshore Oil Platform A

At this point, the system and algorithms have been detailed, and it is time to put those concepts into practice. So, this chapter introduces the methodology developed and the results of using it on the data coming from the offshore oil Platform A[1].

In Section 4.1 we define, among a set of faults of interest, which ones are relevant in Platform A, and in Section 4.2 we determine which tags and wells are available to model the previously chosen fault. Section 4.3 details the steps taken to get the data ready to be input to the classifier, and Section 4.4 brings the results obtained in this platform. Section 4.5 closes the chapter with a summary followed by some observations.

4.1 Defining the Failures of Interest

In Chapter 2, we exposed the relevance of addressing failures related to the well's operation. In this work, we restrict ourselves to a particular set of possible faults:

1. Hydrate in the injection/production line
2. Hydrate/clogging in gathering equipments
3. Hydrate/clogging in service line
4. Inorganic incrustation in injection/production column
5. Inorganic incrustation in injection/production line
6. Organic incrustation in injection/production line
7. Paraffin in the injection/production line

[1]Unfortunately, it is necessary to use an alias for some details of the data, to preserve the confidentiality of the data and of the company that provided it.

It is worth mentioning that this split according to *types of faults* is made based on the Production Loss Database, once again. These failures are different instances of the column *Sistema* (Portuguese for System), see Section 2.1.3.

So, this work aims at studying the behavior of hydrate, paraffin, and incrustation related failures. In a machine learning context, it is somewhat difficult to design a system to model a failure using only a few occurrences of it. Thus, it is necessary to narrow even more the failures, retaining the ones with a reasonable number of occurrences. As expected, this analysis needs to be made on a per-platform basis, considering that each platform has its specific fault behavior. In Figure 4.1 there is the distribution of failures according to the System. Platform A has only two target failure types, and one of them occurred only four times, compromising its use.

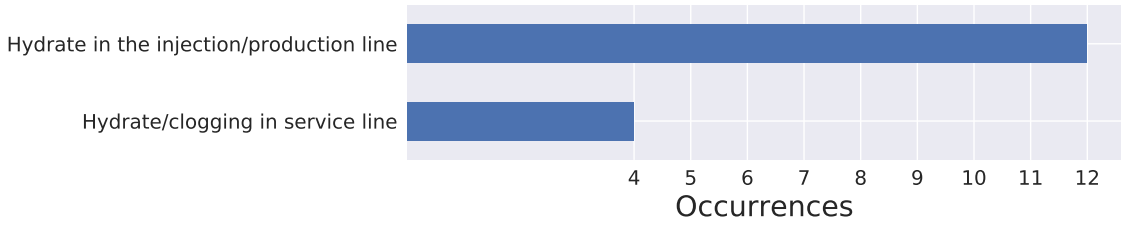


Figure 4.1: Number of events of each failure type of interest in Platform A.

However, from the point of view of the company that manages the platforms, the goal is to minimize the loss by preventing these failures. In Figure 4.2, the distribution of the total loss related to the faults of interest is presented, in percentage. From this figure, one notices how the hydrate in injection lines is also much more relevant regarding the loss than the other fault types.

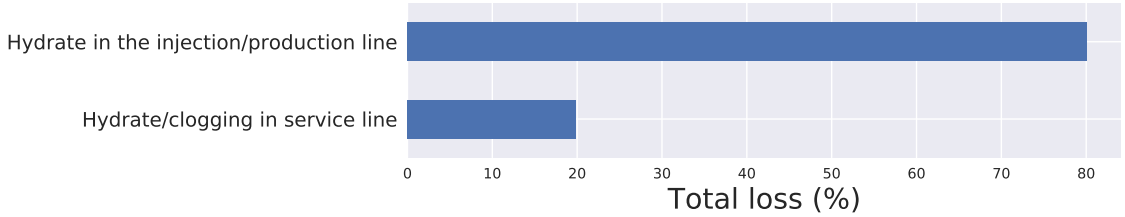


Figure 4.2: Distribution of the total loss according to the types of faults of interest in Platform A.

Even though there are probably too few events to develop a model, the purpose of using this data is to develop a methodology that can be replicated in other platforms. This selection of a target fault is the first step in this methodology, and the next one is the well and tags selection, detailed below.

4.2 Wells and Tags

The way the problem is approached in this work is by considering all wells independent of each other. This means that if a failure occurs in more than one well, it generates more than one event. Designing a model that succeeds independently of the well that the data belongs to is very desirable, since it means that the model has good generalization capabilities. Figure 4.3 reflects the events distribution according to the available wells, and one can notice that it would be rather hard to build a model for each well separately.

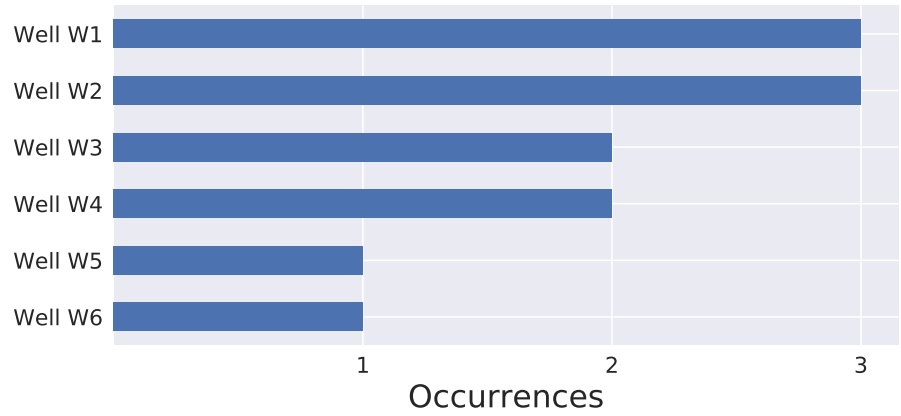


Figure 4.3: Number of fault occurrences by wells. The wells are represented by codes in order to preserve the company data.

Another point discussed in Chapter 2 is the number of tags related to a given platform. In the case of tracking only a specific set of failures, it is useful to remove tags that are not related to the area of study.

The approach of modeling only one system for all wells also limits the number of tags, once it is necessary to have similar tags for all wells. For example, a tag which description is "Well W1 instant temperature" can only be used if the tags that measure the instant temperature of the other five wells are also available. In terms of data structure, Figure 4.4 shows how the final data is composed from the instances of each well.

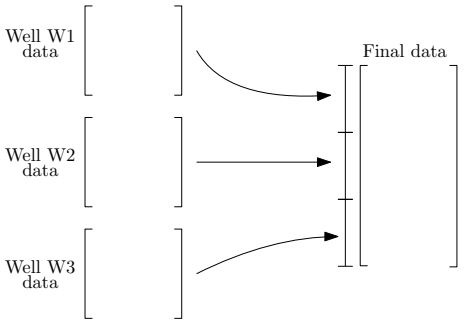


Figure 4.4: The database is simply the vertical concatenation of each well database.

Another characteristic of the received data is the non-uniformity of its beginning and end; there are cases of a tag having entries for only 2 months. Thereby, it is necessary to verify if a tag is indeed available during all fault occurrences in the well that this tag belongs, guaranteeing that all faults are representative. This approach considers the number of occurrences more critical than the number of tags available since it maintains the number of events by eventually putting some tags away.

Fortunately, in Platform A there were no tag problems, as all tags were available for all wells during their respective faults. Table 4.1 brings the list of tags that will be used in this work, alongside its description and symbol used to reference it.

Table 4.1: Description of each tag used by the system developed for Platform A.

Tag description	**Symbol**
Daily closure	t_1
Choke opening	t_2
Header gas lift	t_3
Total gas lift	t_4
Downstream choke pressure	t_5
Gas lift temperature	t_6
Gas lift	t_7
Upstream well temperature	t_8
Further upstream pressure	t_9
Upstream gas lift pressure	t_{10}
Downstream gas lift pressure	t_{11}

4.3 Preparing the Data

At this point, the data is already selected, but it is still necessary to clean it. The received data is the measurement output, meaning that it comes directly from the OSI Soft Pi System, and then, it is necessary to remove the correcting error codes.

The first step is to guarantee that the data is numerical. The approach adopted to define if a tag is numerical is by determining the percentage of strings among its entries, and then, if this percentage is greater than a fixed number, this tag is considered categorical. In this work, we consider as numerical tags the ones that have less than 10% of their entries as strings. Figure 4.5 brings the distribution of the ratio of strings on each tag of Platform A, and one can conclude that the majority of tags is numerical under the criteria adopted in this work.

After selecting only numerical tags, it is necessary to remove the symbols from them. By doing this, more missing data is created, raising the need for interpolating the data to fill in the gaps. However, before interpolation, a minor step is needed: removing outliers.

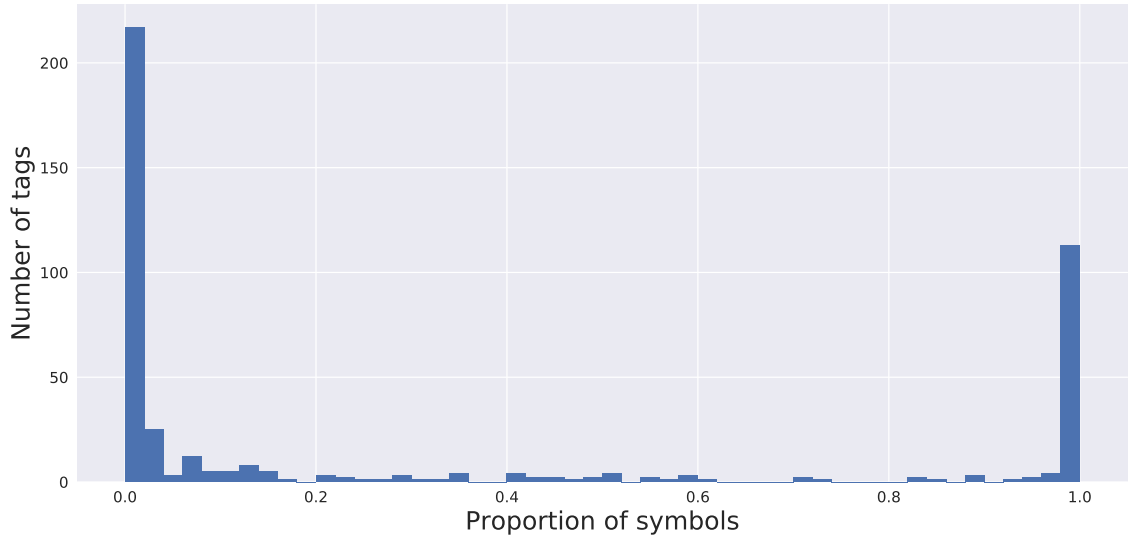


Figure 4.5: Histogram of the proportion of symbols in a tag.

During operation, a sensor can register its measurement as a number that does not reflect the reality. Usually, this number is too large or too small, making it easier to interpret as an error. Thus, those values are removed by establishing a threshold; if an entry is higher than it, then this sample will be removed from the data. A more robust method to remove outliers may be still necessary, but this first removal step is mandatory, as these large numbers deteriorate the data interpolation. Figure 4.6 shows the number of spurious values in the tags related to Platform A, in which there are only a few occurrences in 21 out of 66 tags. Despite the low frequencies of these spurious values when compared to the over 10^6 entries per tag, this step is vital.

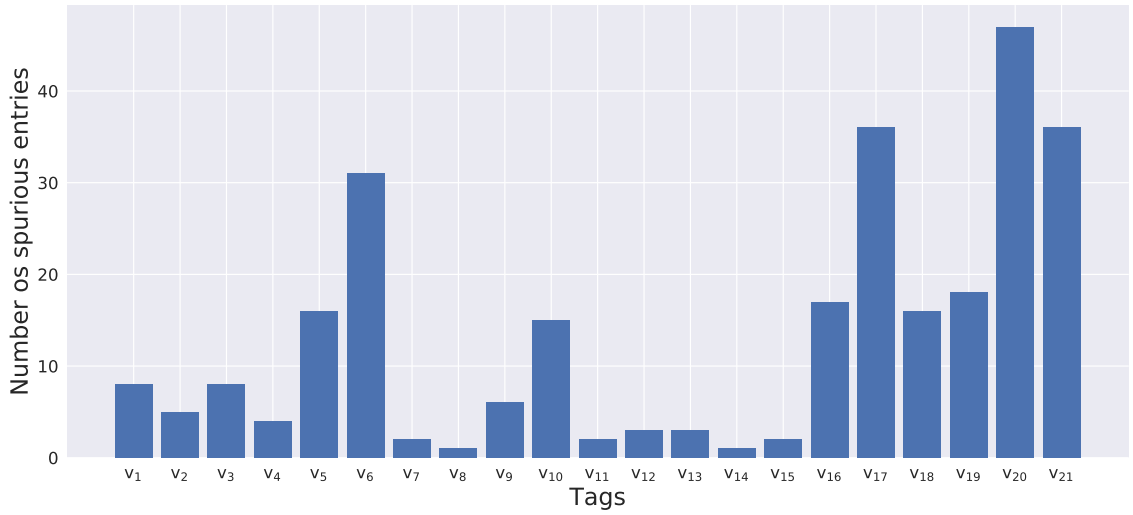


Figure 4.6: Illustration of the number of spurious values removed from each tag v_i out of the 66 selected tags on Platform A. We omitted the other 45 tags, since they had no spurious values.

The next method applied is known as Tukey method and uses the interquartile range to identify outliers [47]. The IQR is defined as the difference between the third and first quartiles, representing the range in which lie 50% of the data that better approximate the data median. Considering this, the Tukey Method considers a sample x as an outlier if the following condition is not met:

$$q_1 - 1.5IQR \leq x \leq q_3 + 1.5IQR, \tag{4.1}$$

where q_1 and q_3 are the first and third quartiles, respectively. The results of applying the interquartile range method on the Platform A data is a removal of 8.93% of samples, on average.

4.4 Experimental Results

This section is composed of two subsections: the first describes the search for the set of hyperparameters that delivers the best result on cross-validation; the second discusses the improvement of using the pre-training operation.

4.4.1 Classifier Results

The Platform A database at this point consists of 12 events of the target fault. So, the model will be trained using group 12-fold cross-validation, which is equivalent to leave one event out of the training set per fold.

The pre-training is not applied in this section, as we want to analyze its impact separately. Thus, first we search through the hyperparameters related to the classifier training, which consists in 760 possible setups according to the following list:

- Feature extraction parameters:
  - Window size: $N \in \{10, 20, \cdots, 90, 100, 200, \cdots, 1000\}$
  - Window step: $s \in \{1, 5\}$
- *Random Forest* parameters:
  - Number of trees (estimators): $B \in \{40, 100\}$
  - Maximum depth: $d \in \{3, 5, 7, 9, \infty\}$
- Other parameters:
  - Balance ratio: $R \in \{1, 2\}$

The results are sorted according to the mean accuracy of the 12 folds ($mACC$). Table 4.2 presents the best five profiles, where *std* stands for the standard deviation associated to mACC, and, $gACC$ is the global accuracy, i.e., the value obtained by summing all folds true positive and true negative values, and then dividing it by the total number of samples. Analyzing Table 4.2, one can notice that the best model according to the mACC is also the model that achieves the highest F_1 score (see Equation (3.14)), making it the best candidate to analyze the other metrics and to evaluate the pre-training effect.

Table 4.2: Top 5 profiles in terms of mACC. The table also brings its performances according to other figures of merit.

R	**N**	**s**	**B**	**d**	**F_1**	**Precision**	**Recall**	**gACC**	**mACC**	**std**
1	500	5	40	3	0.7556	0.797	0.7184	0.7683	0.7824	0.1122
1	800	5	40	3	0.7546	0.7832	0.728	0.7635	0.7789	0.1093
1	400	5	40	5	0.7406	0.8085	0.6833	0.7614	0.7776	0.1161
1	1000	5	40	3	0.7462	0.7781	0.7169	0.7564	0.7716	0.1253
1	100	5	40	3	0.7309	0.7968	0.6751	0.7522	0.7491	0.1174

To analyze the behavior of the classifier during the events, we plot two tags[2] along the event duration. Figure 4.7 illustrates the classifier response during an event and a period before and after it, to study its behavior around the failure. Before taking any conclusions, one can see that the classification is noisy, which makes it hard to interpret the figure.

This indicates the need to perform a final post-processing step, which enforces a temporal consistency to the classifier output. In this process, at every window of T samples the most common output is considered to be the classifier output of the whole group. The window slides in steps of T samples. One can notice that this process adds a delay of T samples to the model output, putting a constraint to the use of large values for T. To better determine this value, Figure 4.8 shows the histogram of the length of the periods composed by consecutive false positives in Figure 4.7.

For Figure 4.8, it is possible to assert that $T = 60$ covers almost all occurrences, and, in the context of hydrate events, it is an acceptable delay. To illustrate the effect of using this post-processing, Figure 4.9 shows the same event after applying the temporal consistency to the classifier output.

This event is a great example of classifier behavior, presenting a high accuracy during the fault, and anticipating it by a good margin.

[2]Both tags have been chosen to make the figures easier to interpret, making the fault interval clearer.

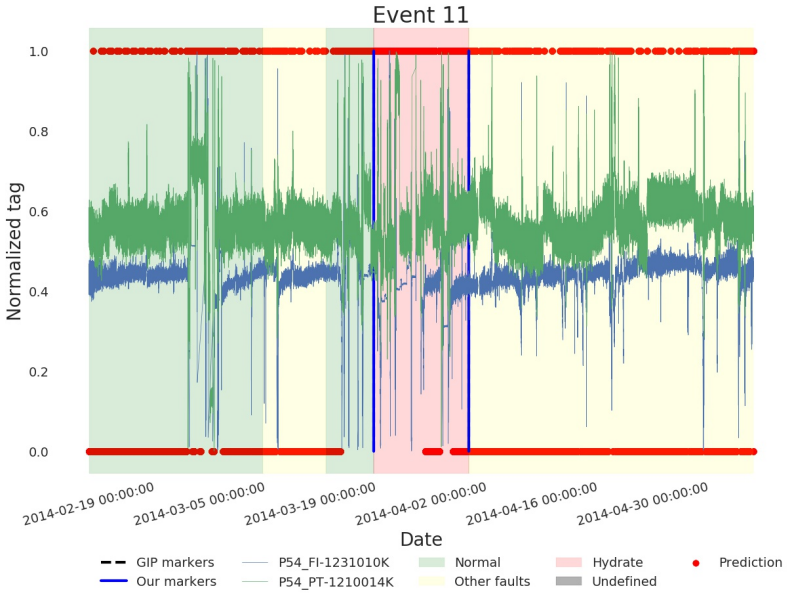


Figure 4.7: The background represents the platform state at the time: green for normal operation, yellow for other faults, red for the target fault, and gray for the period elapsed between ours and PLD markers. The red markers represent the classifier response over the time, which assumes a value equal to 1 when it detects a fault and 0 for normal operation. Two tags are also plotted over time, for visualization purpose.

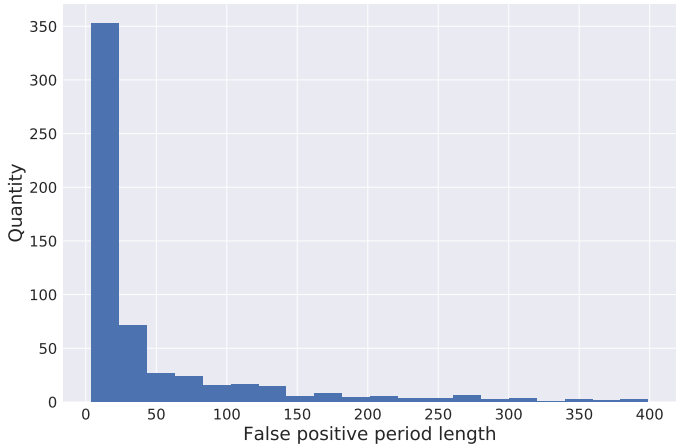


Figure 4.8: Analysis of the false positive length. Each time a false positive occurs repeatedly, the number of consecutive samples represents its length.

4.4.2 Pre-training Results

Pre-training is not performed during the classifier hyperparameters search. This is so because adding the pre-training block to the classifier development increases its overall computational cost tremendously. This growth is even more substantial when applying multiple rounds of pre-training. So, searching for the best set of pre-training hyperparameters is impracticable, once the searching would take 100 times the current execution time.

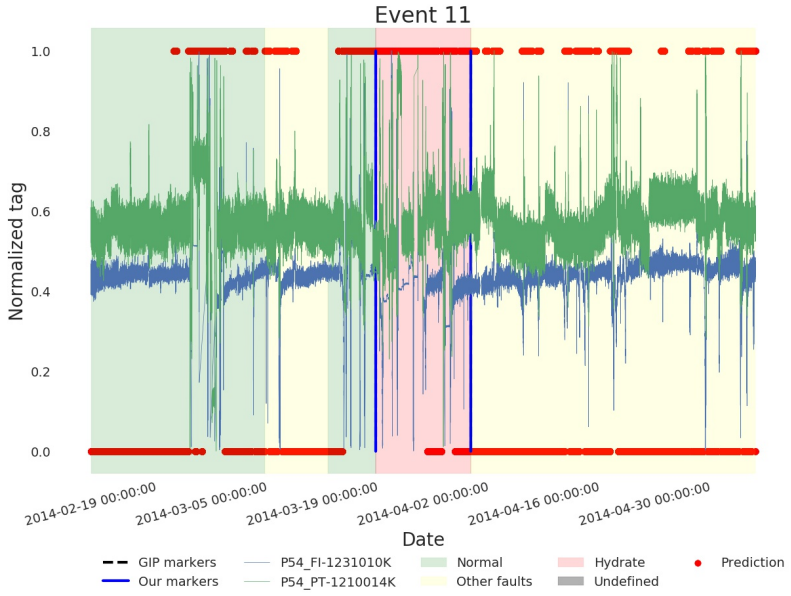


Figure 4.9: The same event plotted in Figure 4.7 after applying the temporal consistency to the classifier output. One can notice that the classifier response is much less noisy.

Another reason that requires the pre-training analysis to be done separately is that its performance has to be analyzed in a qualitative manner, as explained in Section 3.4.

In the previous section, we found the model with the best performance, and the pre-training model will be evaluated considering this model. The first result to be analyzed is represented in Figure 4.10, which brings all the possible combinations of parameters. One can claim that the curves present the same behavior, increasing its performance as the value of τ increases. This increase in τ means that the system becomes more selective when discriminating the normal samples, proving that the algorithm is working as intended.

It must be emphasized, however, that it is still necessary to address the generalization capability of the pre-training procedure when dealing with other normal samples. This matter is discussed through the results on the extra validation set, illustrated in Figure 4.11. In this plot, two values are seen: the maximum accuracy obtained by a model using a given number of rounds, and the accuracy of this same model in the extra validation set. This figure presents the trade-off for increasing the number of rounds: it improves the overall accuracy but makes the model much more selective when dealing with general normal samples. This decreasing performance was expected, since the extra validation set has normal samples that have not been submitted to the cleaning process. Thus, when the model evaluates these samples, it will classify a portion of normal samples as faults, decreasing the model accuracy.

So, picking the optimal parameters is a combination of interpreting the trade-off

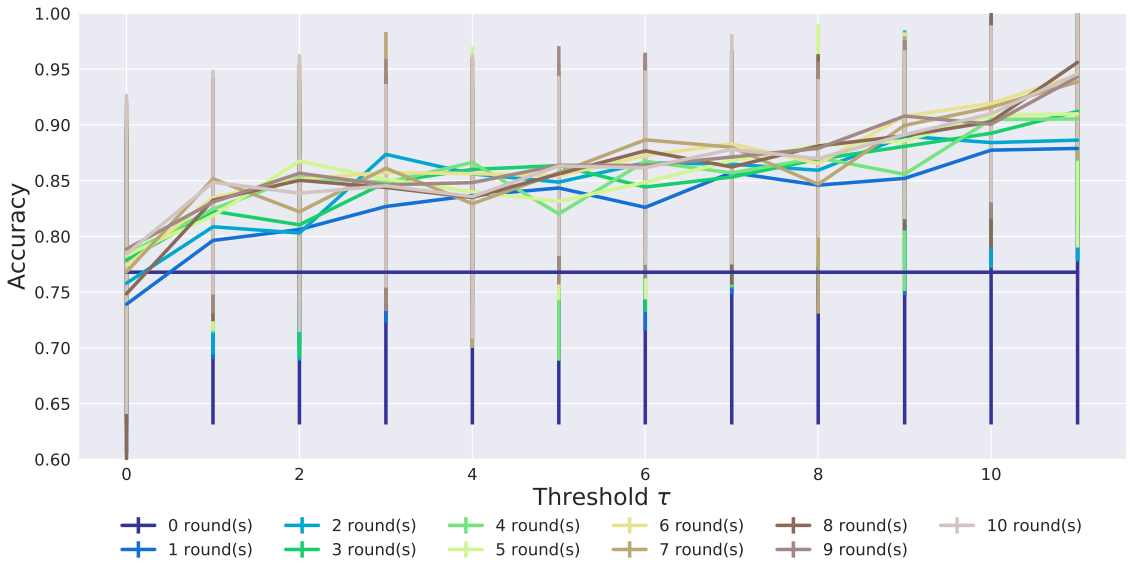


Figure 4.10: Illustration of the results for all pre-training hyperparameters combinations. One can notice an apparent tendency of getting higher values for accuracy as the threshold τ increases, despite the number of rounds.

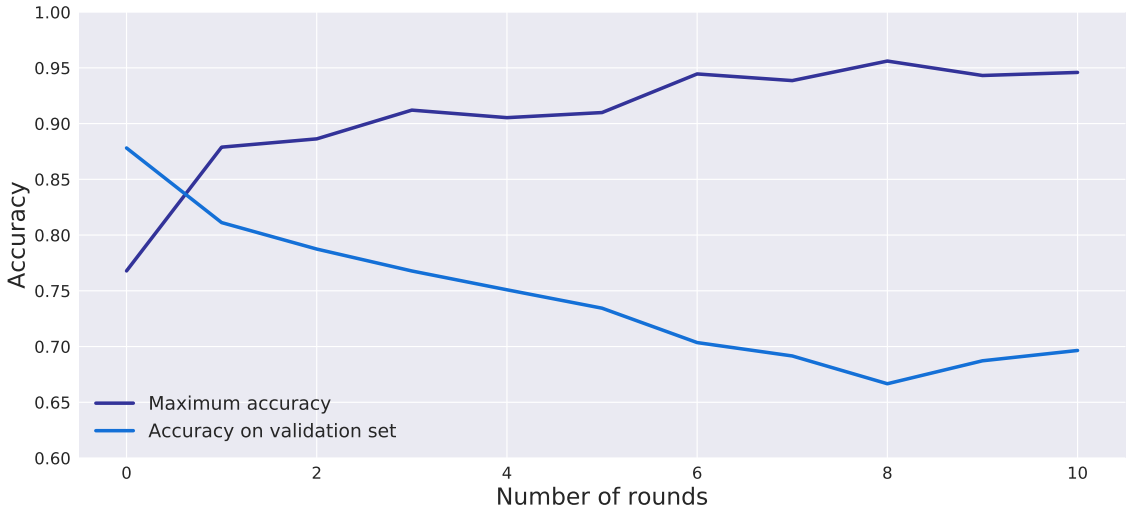


Figure 4.11: Illustration of the trade-off for increasing the number of rounds of pre-training method: although it improves the in-sample results, it decreases the accuracy in the extra validation set.

mentioned above with the knowledge of the data domain. If it is expected that the normal samples are utterly contaminated, getting 60% of accuracy on the extra validation set is not as bad as it sounds.

In the context of this work, the decrease of accuracy on the extra validation set caused by using one round of pre-training is justified by the increase on the overall classifier accuracy.

In what follows, there is a summary of the results obtained studying Platform A as detailed in the present chapter:

- Hyperparameters of the best model:
  - $R = 1$
  - $N = 500$
  - $s = 5$
  - $B = 40$
  - $d = 3$
- Pre-training hyperparameters:
  - 1 round
  - $\tau = 11$
- Mean accuracy: 0.879 ± 0.1
- F-1 score: 0.864

4.5 Conclusions

Handling the real data has helped a lot in the development of the methodology that will be further applied on the next offshore oil platforms. It took approximately 6 months to come up with the final results, since the first contact with the data, and it is expected that this time decreases significantly for the next platforms analyzed.

The results are also satisfactory, reaching 88% of accuracy using a model that is considered simple, demanding a derisory amount of computational resources to run once it is properly trained.

Chapter 5

Results for Offshore Oil Platform B

In the previous chapter, we analyzed the data from Platform A. The process started by determining which faults of interest suit the problem in that platform. Following, we determined which wells and tags are available to compose the platform dataset. Thereafter, we cleaned the data and made the final preparatives to get the data ready for the system. In the present chapter, all these steps will be replicated on Platform B data.

Once the data is cleaned, the system will be trained, and the results and discussions about the system performance on Platform B will be presented during the last sections of this chapter.

5.1 Defining the Failures of Interest

The target failure needs to be chosen according to the loss associated to it and the number of occurrences on the platform. In Platform A the predominant fault during the observed period is "Hydrate in the injection/production line". Figure 5.1 exhibits the fault distribution for Platform B, indicating that the same type of fault has the most significant influence on the total loss. However, there is not a single prevailing type of fault in Platform B; one can state that the first three types of fault are relevant and justify the efforts to analyze it:

1. "Paraffin in the injection/production line"
2. "Organic incrustation in injection/production line"
3. "Hydrate/clogging in service line"

From the machine learning point of view, however, a problem is only relevant if there is enough data to study it. Therefore, it is necessary to check the number of occurrences of each type of fault in order to evaluate the feasibility of using a machine learning algorithm in it.

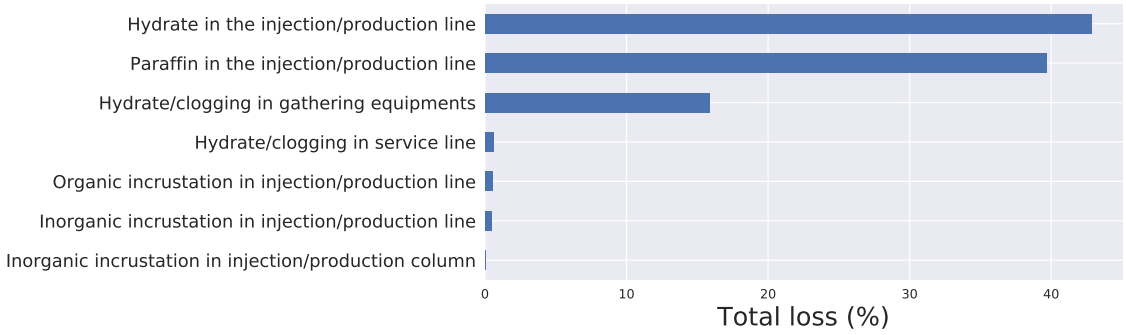


Figure 5.1: Distribution of the total loss according to the types of faults of interest in Platform B.

Unfortunately, by analyzing Figure 5.2 one can conclude that is only possible to model one of those three types of faults. Even though a single event of Hydrate in injection/production line corresponds to more than 40% of the total loss; it is most unlikely that this type of fault can be generalized from a single example.

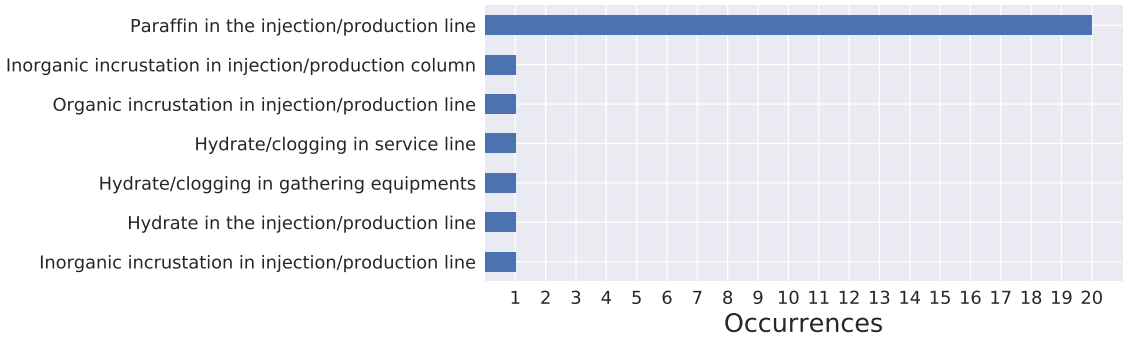


Figure 5.2: Number of events of each failure type of interest in Platform B.

Therefore, in Platform B the only type of fault that meets both criteria of significant loss and sufficient number of events is "Paraffin in the injection/production line". This scenario is an excellent opportunity to confirm that the methodology devised in Chapter 4 is not restricted to a specific type of fault.

5.2 Wells and Tags

Once the target fault has been identified, it is necessary to check how the events are spread according to the wells. In Figure 5.3 there is the paraffin event distribution according to which wells the fault occurred, showing that each event occurs in only one well at a time.

In Platform A there were only 12 events available, which may have restricted the generalization capability of the model. In the current platform, there are 20 events, distributed across only two wells, to train the model. However, the increase on the

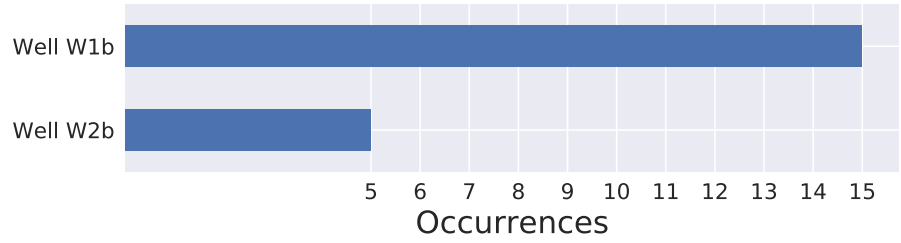


Figure 5.3: Distribution of the fault occurrences according to the wells.

number of events is not enough to balance the classes (normal operation and target fault): in Well W2a, for instance, 82.2% of the data represents the normal operation, where 11.8% is the target fault, and the other 6% is related to other faults. The data is even more unbalanced in Well W2b, where only 0.75% is the target fault, the normal operation covers 98.66% of the samples, and 0.59% represents other faults. As described in Section 3.5, we address this question by sampling the normal data to have the number of target fault samples multiplied by the factor R.

The next step in the system development is guarantee that the data is numerical. The approach used in this work to define which tags are numerical is straightforward: if the proportion of symbols of a given tag is greater than a fixed threshold, then this tag is considered categorical. Figure 5.4 illustrates the distribution of tags according to the proportion of symbols. If one considers a tag being categorical if its proportion of symbols is greater than 0.9, it is possible to claim that there are more categorical tags than numerical in Platform B.

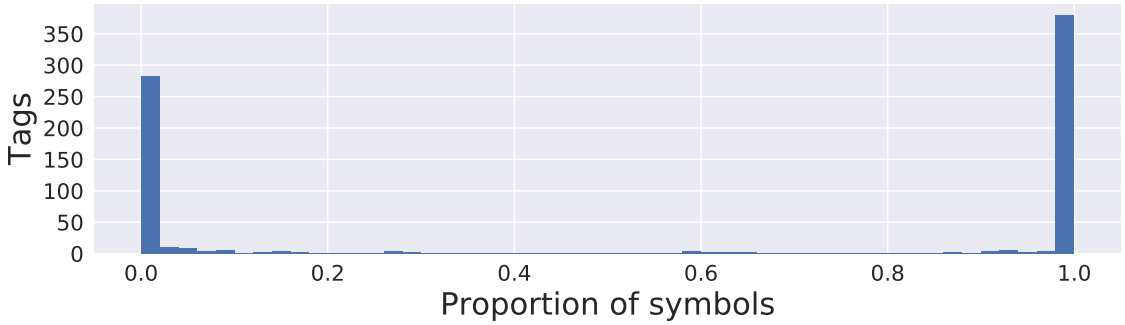


Figure 5.4: Histogram of the proportion of symbols in a tag in Platform B.

Now, dealing only with numerical tags, it is necessary to restrict them to the ones that contemplate all the fault intervals. In Figures 5.5 and 5.6 we present histograms of the start of each tag for each well. Each figure also contains vertical dashed lines representing the start[1] of the faults that occurred on the respective well[2]. It is possible to notice that there are a few tags that start after some faults,

[1]By start we mean the first non-empty entry on that tag value.

[2]Since the duration of all faults is really short compared to the duration of the tags, we decided to omit the end of each fault to keep the figures clearer.

so they were removed from subsequent processing. Fortunately, all tags end after all fault periods, and then there is no need to discard any tag due to this criterion.

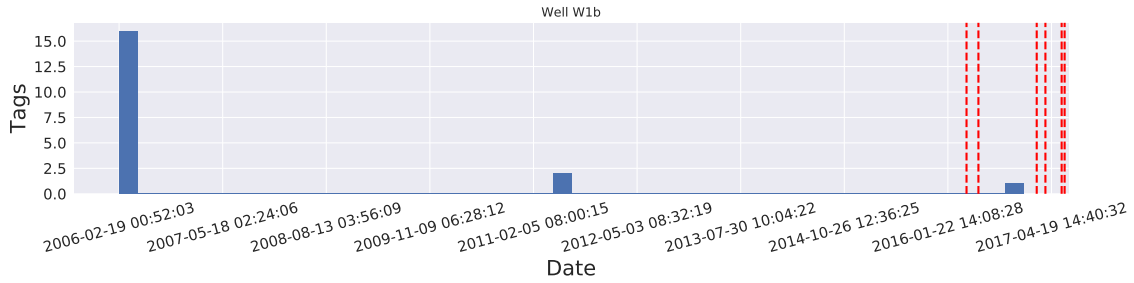


Figure 5.5: Histogram of the start of the tags on Well W1b.

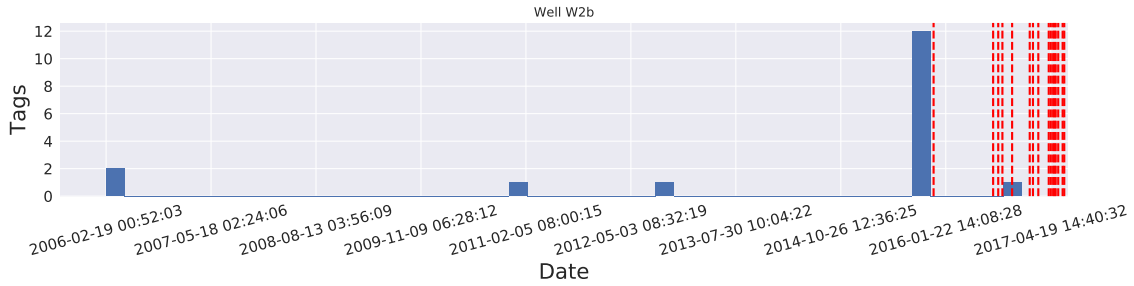


Figure 5.6: Histogram of the start of the tags on Well W2b.

The last step before preparing the data is defining which tags are available for both wells. Table 5.1 shows which tags will be considered in this platform, where one can notice that the majority of tags are related to pressure sensors.

Table 5.1: Description of the tags employed on the analyzes of Platform B.

Tag description	**Symbol**
Gas lift injection rate	t_1
Production choke opening	t_2
Launcher downstream pressure	t_3
Pressure inside the ring riser cover	t_4
Upstream pressure on launcher	t_5
Choke upstream pressure	t_6
SDV upstream pressure	t_7
Pressure inside production riser cover	t_8
Pressure inside Christmas tree equipment	t_9
Ring pressure	t_{10}
Pressure on production column	t_{11}
Choke upstream temperature	t_{12}
Temperature inside Christmas tree equipment	t_{13}
Temperature on production column	t_{14}

Unfortunately, the set of tags of this platform is considerably different from the selected tags on Platform A, rendering unfeasible the use of a single model for both

platforms. At this point, one can perceive the value of establishing a methodology that can be easily reproducible and scaled to other types of input data.

5.3 Preparing the Data

First, we removed the spurious samples, and then we applied the Tukey Method. The first step only affected the tag "Temperature on production column" related to Well W1b, and 56 instances were removed. The second, and last step proved to be more efficient, it removed 7.2% of the samples considered as outliers.

When comparing these results with the ones obtained on Platform A, one can conclude that Platform B data is cleaner, presenting less spurious values and considerably less outliers. This reinforces the value of the methodology and shows that different results are expected during every phase of the analysis, even though the datasets have the same recording process.

5.4 Experimental Results

5.4.1 Classifier Results

In Platform B, we decided to explore more values for the window step. In Platform A we only tested two values for it: 1, once it is its minimum value; and 5, because as we increase the step, we decrease the number of samples, and since target faults sample is an important resource in those datasets we cannot use high values. So, the search routine goes through 1900 combinations of parameters:

- Feature extraction parameters:
  - Window size: $N \in \{10, 20, \cdots, 90, 100, 200, \cdots, 1000\}$
  - Window step: $s \in \{1, 2, 3, 4, 5\}$
- *Random Forest* parameters:
  - Number of trees (estimators): $B \in \{40, 100\}$
  - Maximum dept: $d \in \{3, 5, 7, 9, \infty\}$
- Other parameters:
  - Balance ratio: $R \in \{1, 2\}$

The results are sorted according to the mACC score, and Table 5.2 presents the best five profiles. Once again, the model that achieves the highest mACC also

achieves the highest F_1 score. So, we selected it to study the effects of pre-training the model. Note that with the exception of the balance ratio (R), all other hyperparameters from the selected model are different from the ones on the model selected in Platform A. The mACC achieved by the set of hyperparameters chosen in the previous platform is 75.87 ± 17.4, which is considerably lower than the 83.59 ± 11.78 achieved for Platform B..

Table 5.2: Top 5 classifier profiles in terms of mACC, along with their performances with other figures of merit.

R	**N**	**s**	**B**	**d**	**gACC**	**mACC**	**stdACC**	**F_1**
1	1000	1	100	5	0.7309	0.8359	0.1178	0.7761
2	1000	1	40	7	0.773	0.8344	0.1496	0.7463
2	1000	3	40	5	0.7794	0.8284	0.14	0.7351
1	1000	3	100	7	0.7671	0.8274	0.1299	0.7685
1	1000	1	100	7	0.7167	0.826	0.1459	0.769

After choosing the model, the next step is to evaluate the size of the window applied in the post-processing step to enforce time-consistency in the classifier output. As explained in the previous chapter, this step implies a delay of T samples once the system is operating online. Considering this drawback, in the case of applying it on Platform B, the gain in accuracy (less than 0.2%) does not justify the delay.

During the fine-tuning of the start and end of each event, we decided to create a new label for the period between PLD markers and ours that is outside the one delimited by the manual markers. Figure 5.7 illustrates the system performance during an event and the period around it, where the gray background corresponds to this new label, named Undefined period. In this example, one can notice that the model classifies the samples right before and after the fault period as non-normal. This mislabeling occurs due to the change in the system behavior that cannot be readily identified by just looking at the tags, and the PLD does not contain enough precision on its data.

5.4.2 Pre-training Results

The approach adopted during cross-validation was the one-out, considering one event as the validation set per iteration. Therefore, in this platform, there are 20 folds, which substantially increases the computational cost of training. The first pre-training proposal considered that we would evaluate each normal sample k times, where k is the number of folds used for training the classifier. As we have 20 folds, it would also increases the computational cost of applying the pre-training, when compared to the Platform A. So, these two sequential increases encourage an adaptation on the pre-training method employed in Platform A.

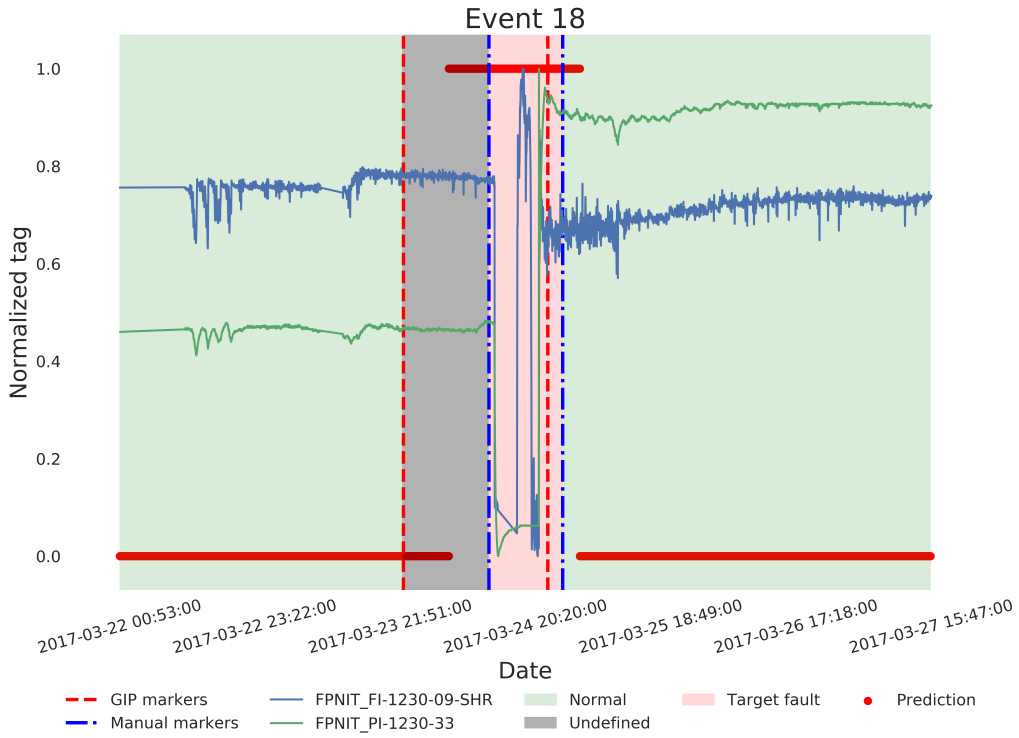


Figure 5.7: Illustration of the model output during one of the events. This classifier was validated on event 18 and trained using the other 19 events.

The way we modified the algorithm for Platform B is by doing only a fixed number of evaluations of a single normal sample. In the previous version, we evaluated it according to the number of folds, seeing that the one-out strategy requires one iteration per event. In contrast to that, for Platform B we did only 9 evaluations, randomly chosen across the $k-1$ times the sample appears on the training set. The results follow the same tendency found on Platform A: as the number of rounds increases, there is a slight increase on performance, but in general, increasing the value of τ is much more effective, as illustrated in Figure 5.8.

To support the choice of the pre-training hyperparameters, it is necessary to evaluate the performance on the extra validation set. Based on Figure 5.9, one can note an unexpected behavior: a single round drastically decreases the accuracy on the validation set; even though the increase on the classifier accuracy is still significant (83.6% to 90.5%), as occurred in Platform A. Despite the boost on classifier performance, it is unlikely that the decrease on extra validation set accuracy is only due to errors during data labeling, since we purged, on average, 13% of the normal samples during the pre-training routine performed in Platform B data. So, along with the mislabeling, the classifier is probably overfitting to a specific set of normal samples. Thus, applying the pre-training method was not useful in this scenario. Following, there is a summary of the results obtained by studying Platform B as detailed in the present chapter:

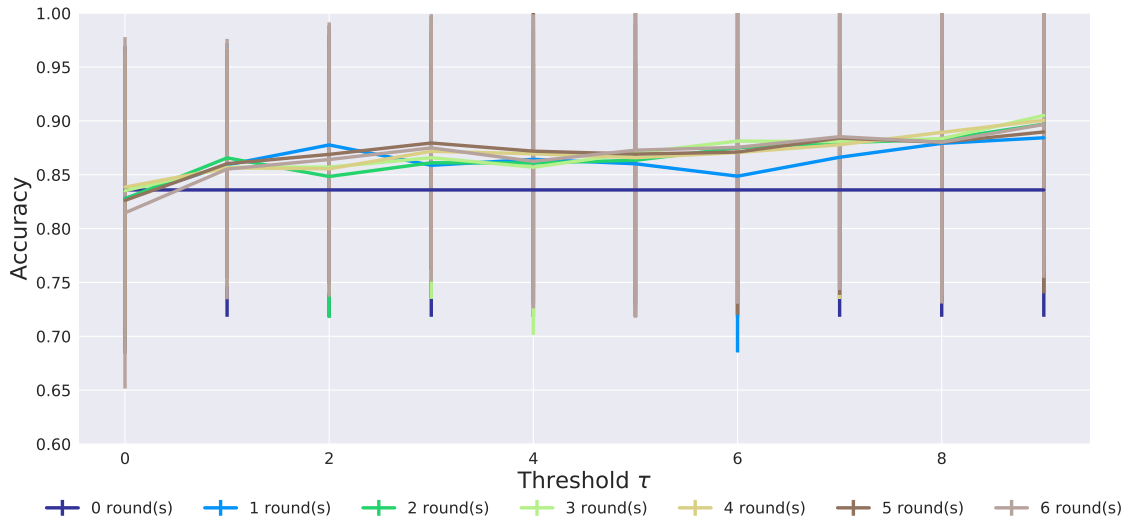


Figure 5.8: Illustration of the pre-training method performance for a variety of hyperparameters. Overall, the accuracy tends to increase as the threshold τ increases, as in Platform A.

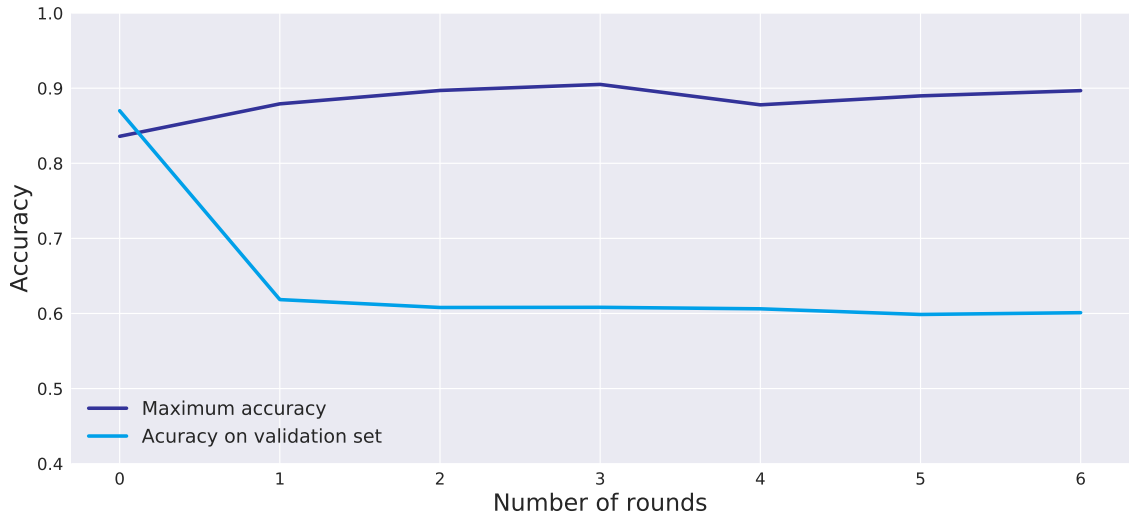


Figure 5.9: Performance of the model using the pre-training method on the extra validation set. In this platform, the increase in the classifier performance is not relevant when weighted by the decrease on the extra validation set accuracy.

- Hyperparameters of the best model:
  - $R = 1$
  - $N = 1000$
  - $s = 1$
  - $B = 100$
  - $d = 5$

- Pre-training hyperparameters[3]: not used
- Mean accuracy: 0.836 ± 0.118
- F-1 score: 0.776

5.5 Conclusions

The methodology developed in the Platform A was replicated on another platform, and the results are also satisfactory, reaching 83.6% of accuracy. The pre-training showed substantial improvement in the classifier accuracy, but due to its performance on the extra validation set, we decided to leave the pre-training out of the training of Platform B. This work demanded six weeks to yield the results presented in this chapter, counting from the first contact with the raw data from PI, which is less than a quarter of the time demanded for analyzing the data Platform A.

Another critical point of this chapter is the fact that we successfully modeled a type of fault different from the one modeled in Chapter 4, considering that the final goal is a system that can handle multiple faults.

[3]In this case, we kept the leave-one-out strategy when training the classifier, yielding 20 folds. However, during the pre-training we used only 10 folds, due to the high processing time it demands.

Chapter 6

Results for Offshore Oil Platform C

This chapter presents the application of the methodology, developed in the last two chapters, on a new database. The Platform C has the same characteristics of the other addressed platforms, but has an advantage: it has 42 occurrences of faults of interest, which is more than double the occurrences in the previous platforms. Along with this, Platform C has more than one representative fault, giving the chance to analyze how the model behaves in the multiclass scenario.

The structure of the present chapter is the following: Section 6.1 defines the faults of interest that are relevant to Platform C; Section 6.2 restricts the data according to the wells attended by the chosen faults; Section 6.3 details the data cleaning; Section 6.4 discusses the model performance during the classifier training and pre-training stages; and, finally, Section 6.5 summarizes the chapter and compares the results against the ones obtained on Platforms A and B.

6.1 Defining the Failures of Interest

When first handling the data received from PI, it is necessary to check which problem is worth to address. This decision is made based on two aspects: the failure must be relevant in terms of production loss to be attractive for the company who provided the data, and the failure must have occurred a reasonable number of times to allow some modeling. This analysis is first delimited by a set of interest faults, which includes:

1. "Hydrate in the injection/production line"
2. "Hydrate/clogging in gathering equipments"
3. "Hydrate/clogging in service line"
4. "Inorganic incrustation in injection/production column"
5. "Inorganic incrustation in injection/production line"
6. "Organic incrustation in injection/production line"

7. “Paraffin in the injection/production line”

In Figure 6.1 we have the contribution of each one of those faults to the total combined loss they produced in Platform C. Following the same pattern of the two other platforms, “Hydrate in the injection/production line” is the predominant fault, reinforcing the relevance of its modeling and detection.

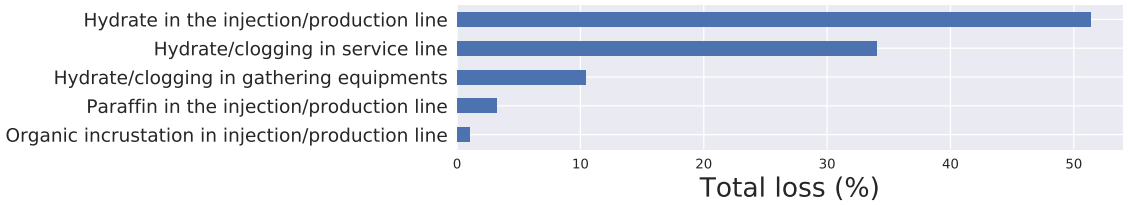


Figure 6.1: Distribution of the total loss generated by the faults of interest in Platform C.

Analyzing the relevance of these faults from a machine learning perspective, one can conclude that there are two faults with a reasonable number of occurrences, as illustrated in Figure 6.2. Based on this, we decided to restrict the analysis to these two types of fault:

1. “Hydrate in the injection/production line”
2. “Hydrate/clogging in service line”

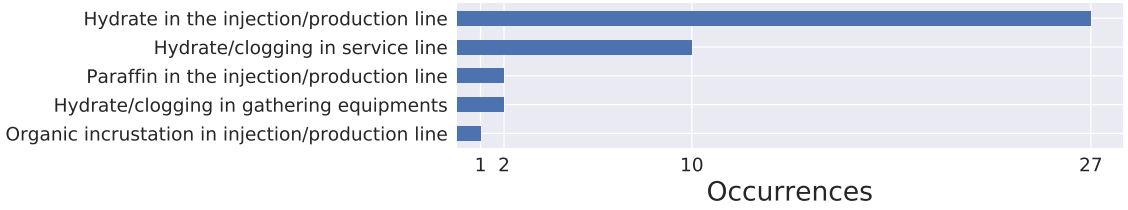


Figure 6.2: Number of occurrences of each fault of interest in Platform C.

In contrast to Platforms A and B, the Platform C dataset has more than one target fault, allowing one to study how the developed methodology responds on a multiclass scenario.

6.2 Wells and Tags

Proceeding in the methodology, we define which set of wells and tags best fit our needs. Figure 6.3 shows how the faults of interest are distributed over the wells, and one can notice that in Platform C there are faults that occur in more than one well at the same time. Consequently, the number of events increases: the 37

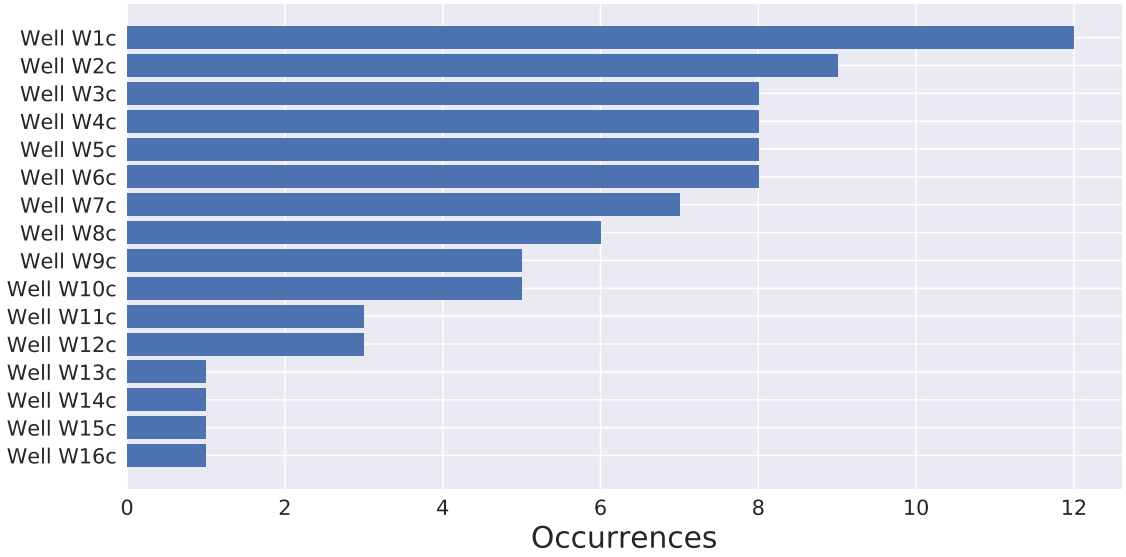


Figure 6.3: Distribution of events according to the wells it occurred. In Platform C many faults occur in more than one well simultaneously, yielding an increase of events.

faults are now represented by 86 events; which is one more particular characteristic of Platform C.

As we are concatenating the data from all wells, it is necessary that all of them have the same set of sensor tags available for processing. Unfortunately, as expected, different wells have different sets of tags available, demanding a search for an optimal selection of wells and tags, which maximizes the number of faults without putting many classes of tags away. Table 6.1 represents this trade-off[1]: as we increase the number of used wells, fewer tag classes will be available on all of them.

Table 6.1: List of tag classes and fault events for all wells. As we consider more wells, eventually a tag class will not be available for one of the considered wells, which automatically discards the class. On the other hand, the number of available events increases, encouraging us to pick as many wells as possible.

Number of wells	**Number of tag classes**	**Number of faults**
1	26	12
2	24	21
3	24	29
4	24	37
5	20	45
6	20	53
7	20	59
8	20	64
9	6	69

[1]In this analysis, we did not consider Well W7b, as it does not contain any numerical tag.

Thereby, based on the Table 6.1 results, choosing the first 8 wells gives 64 events of the target faults and restricts the number of classes to 20, which is more than there are in both previous cases added together. To guarantee that all these classes can be used, it is necessary to verify if they are numerical and if they cover all faults duration.

Fortunately, all of those 20 classes are numerical, even though Platform C presents the same pattern observed on the previous platforms: the majority of tags are numerical, as illustrated in Figure 6.4.

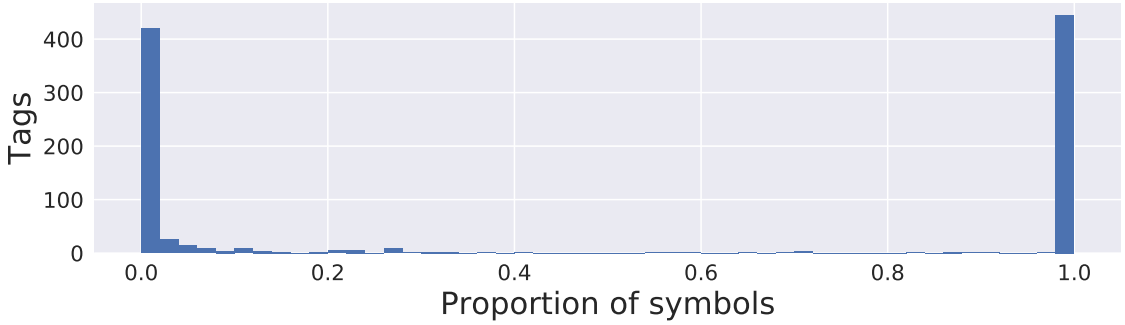


Figure 6.4: Histogram of the proportion of symbols in a tag in Platform C.

Regarding the end of tags, every tag ends after the faults that occurred on the well that tag belongs to. Analyzing the start of the tags, Figures 6.5 to 6.11 present a histogram of the start of each tag for each well,x and the start of every fault that occurred on the respective well. Based on these plots, to cover all faults, it is necessary to remove some tags and all other tags that share the same class.

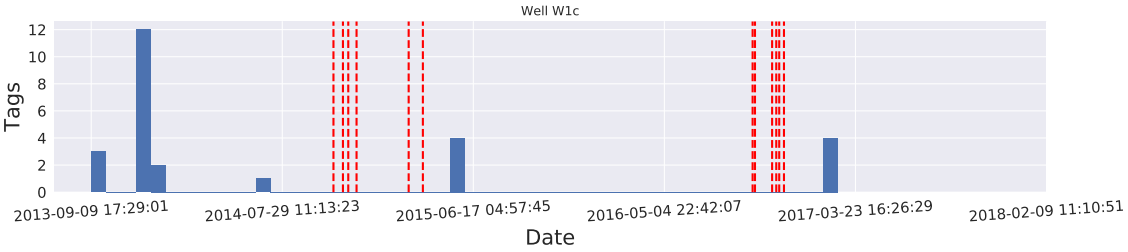


Figure 6.5: Histogram of the start of the tags on Well W1c. The red vertical lines represent the started of an event on this well.

Removing the undesired tags, we ended up with 16 out of the initial 20 classes of tags. Table 6.2 contains the description of each one of them, and once again one can verify that 10 out of 16 classes did not appear on Platforms A and B. This mismatch on classes across platforms also endorses the need for a robust methodology that performs well under different data conditions.

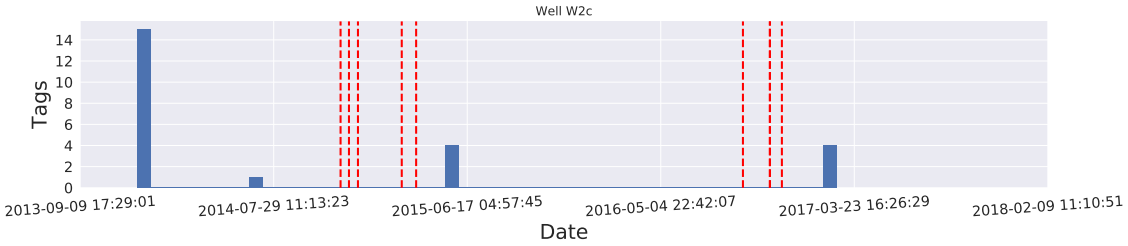


Figure 6.6: Histogram of the start of the tags on Well W2c.

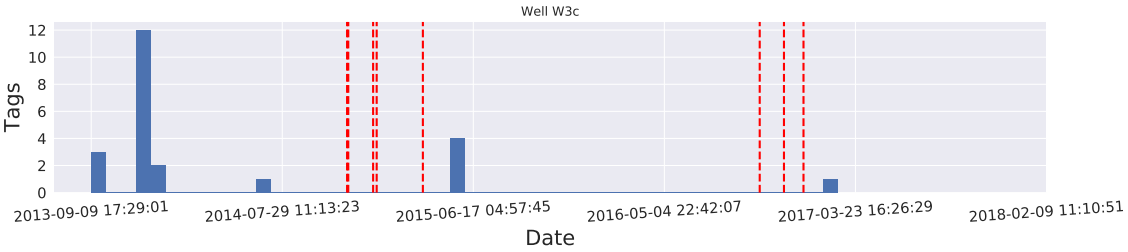


Figure 6.7: Histogram of the start of the tags on Well W3c.

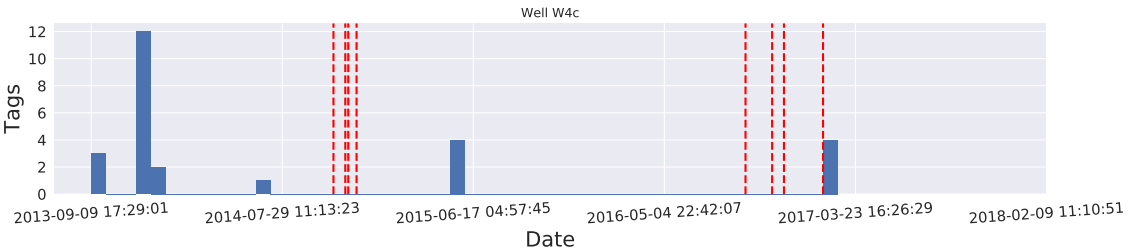


Figure 6.8: Histogram of the start of the tags on Well W4c.

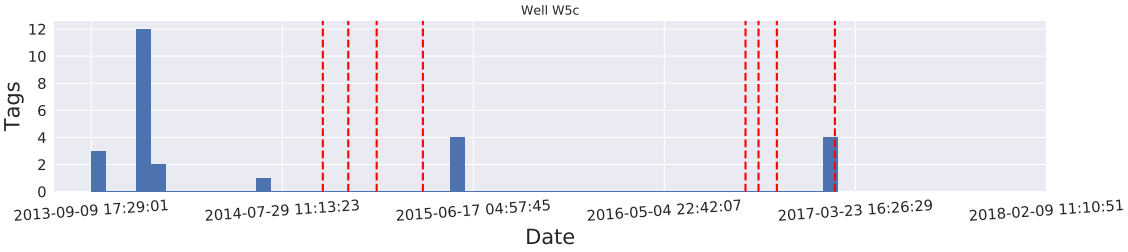


Figure 6.9: Histogram of the start of the tags on Well W5c.

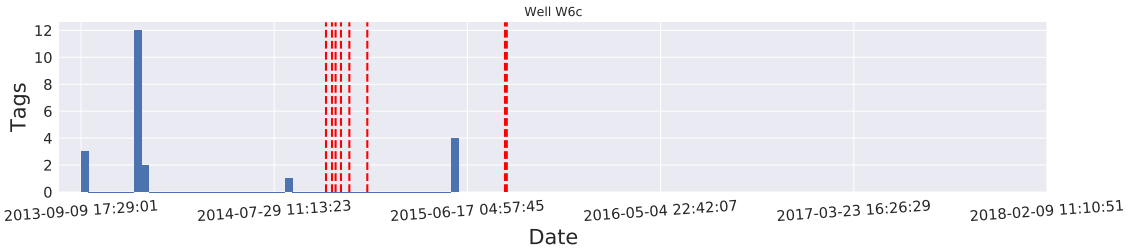


Figure 6.10: Histogram of the start of the tags on Well W6c.

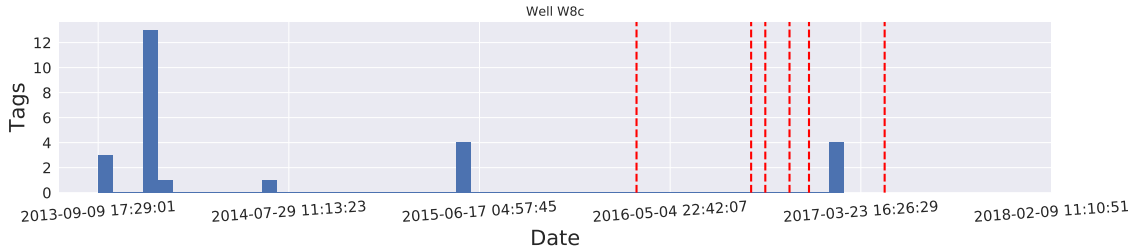


Figure 6.11: Histogram of the start of the tags on Well W8c.

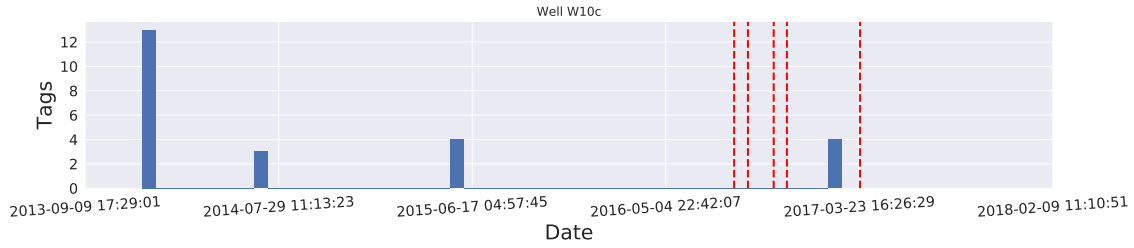


Figure 6.12: Histogram of the start of the tags on Well W9c.

Table 6.2: Description of the tags employed on the analyzes of Platform C.

Tag description	**Symbol**
Corrosion coupon	t_1
Gas lift injection rate	t_2
Gas lift choke	t_3
Pressure on MSGL service line	t_4
Gas lift downstream pressure	t_5
Gas lift upstream pressure	t_6
Delta P	t_7
PDG pressure	t_8
TPT pressure	t_9
Choke opening	t_{10}
Production choke upstream pressure	t_{11}
Production choke downstream temperature	t_{12}
Gas lift downstream temperature	t_{13}
Gas lift upstream temperature	t_{14}
PDG temperature	t_{15}
TPT temperature	t_{16}

6.3 Preparing the Data

At this point all the classes are numerical, and this step consists of removing their outliers. First, we go through all 128 tags (16 classes and 8 wells) searching for spurious values. Figure 6.13 shows that only 7 tags had such occurrences, where one of them had 959 entries considered as anomalies. Next, we applied the interquartile range method to remove outliers before interpolating the data. This step removed, on average, 4.3% of the samples of a tag, which is considerably less when compared

to other platforms (8.93% on Platform A and 7.2% on Platform B).

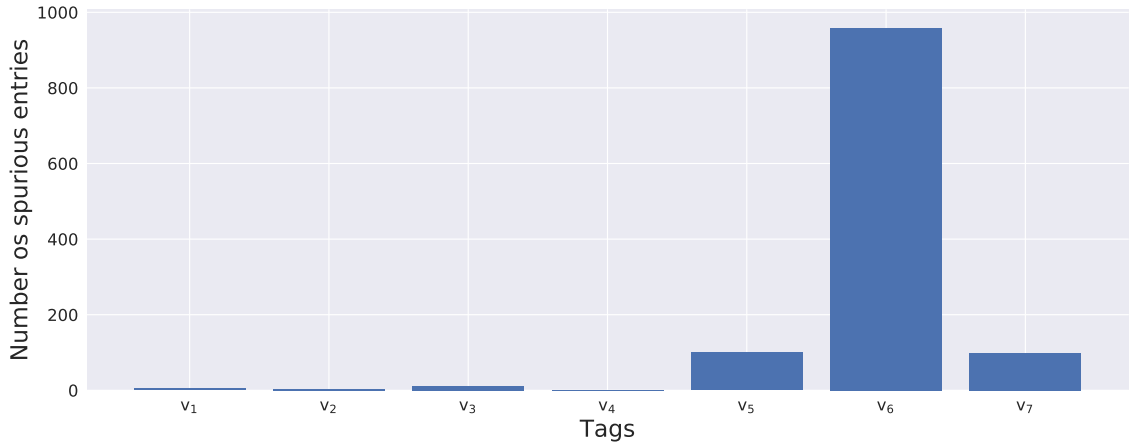


Figure 6.13: Number of spurious entries on each tag used on Platform C. The other 121 tags had no spurious measures.

6.4 Experimental Results

6.4.1 Classifier Results

The sweep through the classifier hyperparameters follows the same schedule applied on Platform B, evaluating a total of 1900 classifier setups:

- Feature extraction parameters:
  - Window size: $N \in \{10, 20, \cdots, 90, 100, 200, \cdots, 1000\}$
  - Window step: $s \in \{1, 5\}$
- *Random Forest* parameters:
  - Number of trees (estimators): $B \in \{40, 100\}$
  - Maximum dept: $d \in \{3, 5, 7, 9, \infty\}$
- Other parameters:
  - Balance ratio: $R \in \{1, 2\}$

In Section 6.1 we concluded that two faults can be modeled in this platform: the fault "Hydrate in the injection/production line" has the majority of occurrences and will be referred to as Fault 1, while "Hydrate/clogging in service line" as Fault 2. Thus, we proposed 4 ways to handle the fault classification problem:

1. Mode 1: Classifies between Fault 1 and normal operation.

2. Mode 2: Classifies between Fault 2 and normal operation.
3. Mode 3: Classifies between any fault and normal operation.
4. Mode 4: Classifies between Fault 1, Fault 2 and normal operation.

Mode 1

This mode is a single class problem trained to classify Fault 1, which is the same target fault of Platform A. Table 6.3 shows the five models that achieved best mACC results; the best model here achieves better results than in the Platform A ($78.24 \pm 11.22\%$), although both lie in the same range, indicating that the model successfully detected a fault in more than one platform. This result has a huge impact, proving the proposed methodology robustness once again.

Table 6.3: Top 5 classifier profiles in terms of mACC in Mode 1, along with their performances with other figures of merit.

R	**N**	**s**	**B**	**d**	$\mathbf{F_1}$	**gACC**	**mACC**	**stdACC**
1	600	3	100	9	0.8112	0.8251	0.8222	0.0789
1	600	5	100	9	0.8114	0.8242	0.8213	0.0739
1	600	3	100	7	0.8096	0.8264	0.8209	0.0948
1	500	3	100	7	0.8103	0.8250	0.8207	0.0859
1	600	5	100	7	0.8057	0.8230	0.8164	0.0948

Mode 2

This is another mode that represents a single class classification problem. However, in this case, we are modeling a first seen fault, even though is hydrate related. Table 6.4 shows that the best combination of hyperparameters reaches 78.22% of mACC, meeting the expectations when dealing with a single class problem. Once again, the methodology generalizes for another type of fault.

Table 6.4: Top 5 classifier profiles in terms of mACC in Mode 2, along with their performances with other figures of merit.

R	**N**	**s**	**B**	**d**	$\mathbf{F_1}$	**gACC**	**mACC**	**stdACC**
2	900	4	100	9	0.7358	0.7786	0.7822	0.1098
2	700	3	100	9	0.7353	0.7747	0.7788	0.1288
2	700	4	100	7	0.7403	0.7734	0.7775	0.1217
2	700	3	100	7	0.7391	0.7714	0.7755	0.1184
2	900	4	40	7	0.732	0.7706	0.7747	0.1035

Mode 3

This mode is also a single class problem, but by the way we formulated it considers both target faults as the same failure, configuring itself as a detection problem. Table 6.5 shows the best mACC results, which represent a significant decrease in performance when compared to the single-class scenarios. In addition to that, the variance also increases, from 11.78% in the worst scenario (Platform A) to 19.55%, which indicates a distinct behavior in this new classification mode. This deterioration in performance indicates that treating both targets as the same classes is not a good strategy.

Table 6.5: Top 5 classifier profiles in terms of mACC in Mode 3, along with their performances with other figures of merit.

R	**N**	**s**	**B**	**d**	$\mathbf{F_1}$	**gACC**	**mACC**	**stdACC**
2	700	4	40	9	0.6547	0.6729	0.6733	0.1955
2	700	5	100	9	0.6509	0.6701	0.6705	0.1864
2	800	3	40	9	0.6477	0.6684	0.6687	0.1883
2	700	5	40	9	0.6461	0.6644	0.6648	0.1805
2	800	5	100	9	0.6439	0.6627	0.6631	0.1864

Mode 4

The last mode considered here consists of the identification of Faults 1 and 2 individually. It is expected that the results of this mode are slightly worse when compared to Mode 3, but Table 6.6 points out an important fact: this mode is much more stable than the previous one, as it presents a smaller variance (30% of reduction comparing the best models variance).

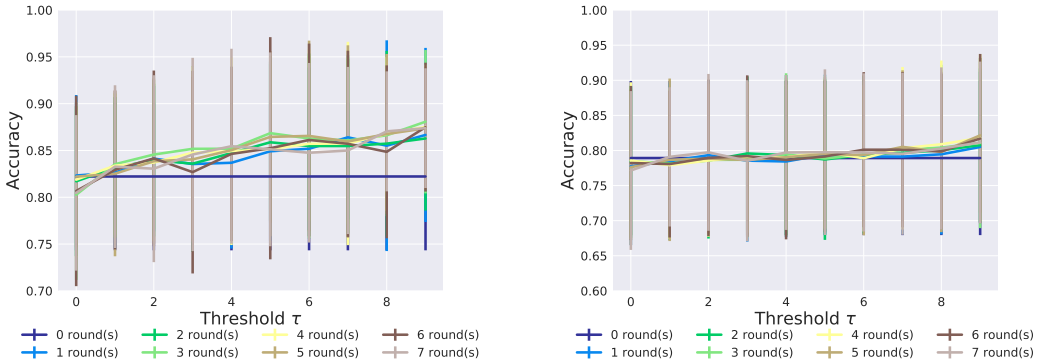
Table 6.6: Top 5 classifier profiles in terms of mACC in Mode 4, along with their performances with other figures of merit.

R	**N**	**s**	**B**	**d**	$\mathbf{F_1}$	**gACC**	**mACC**	**stdACC**
2	900	4	40	9	0.6698	0.653	0.6533	0.1394
2	900	4	100	9	0.6608	0.6465	0.6468	0.1279
2	800	3	100	9	0.6564	0.6425	0.6429	0.1808
2	900	5	40	9	0.6537	0.6423	0.6426	0.1692
2	900	3	40	9	0.65	0.6388	0.6391	0.1686

6.4.2 Pre-training Results

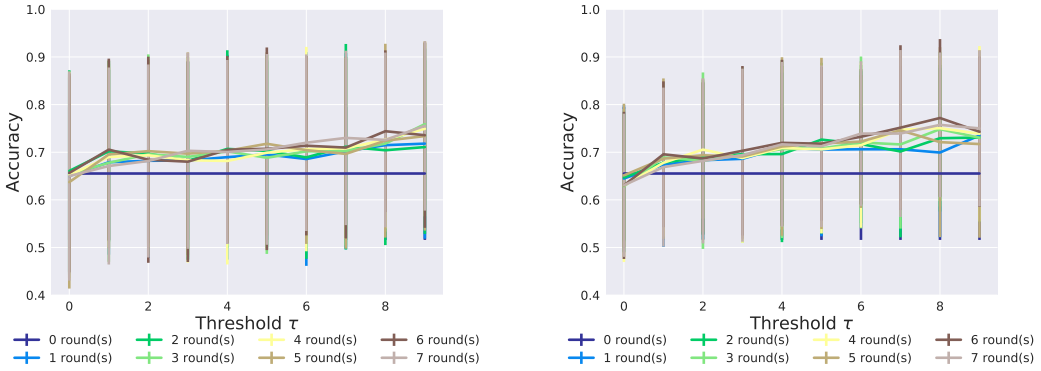
We apply once again the pre-training method detailed in Section 3.4 as one of the steps during model training. Figure 6.14 shows the results for multiple rounds and

different selection thresholds. From these results, one can notice that the same pattern persists: generally, the accuracy increases as the selection threshold increases.



(a) Mode 1

(b) Mode 2



(c) Mode 3

(d) Mode 4

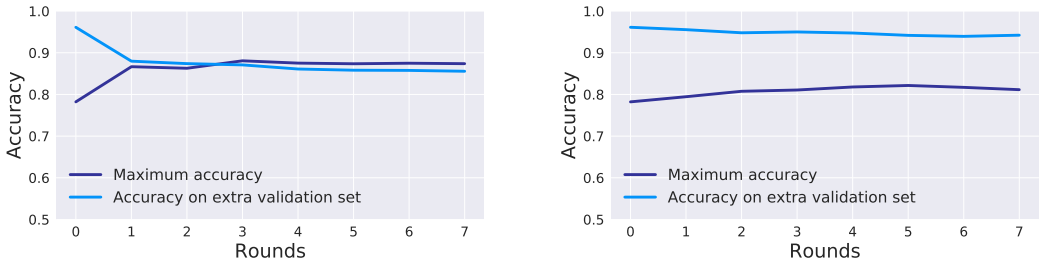
Figure 6.14: Illustration of the pre-training performance for a variety of parameter for the four classification modes. The tendency of having better results for higher thresholds presend in the previous platforms is also observed in all cases once again.

In Platform A the pre-training presented a boost of 12.3% in the best model mACC without pre-training, while the results on Platform B did not justify the use of the technique. To support the selection of the pre-training hyperparameters, Figure 6.15 brings the results on the extra validation set for the models that achieved the best accuracy on training for each number of rounds, where one can observe that:

- Mode 1: In this case, the accuracy on extra validation set does not suffer significant decrease beyond one round, allowing one to choose the set of hyperparameters that delivers the best accuracy on training: 3 pre-training rounds and selection threshold equals to 9.
- Mode 2: This mode behaves precisely as Mode 1 and with that we choose 5 pre-training rounds and selection threshold equals to 9.
- Mode 3: The best result is achieved using 3 pre-training rounds, and selection threshold equals 9. The accuracy on extra validation set decreases 7% but

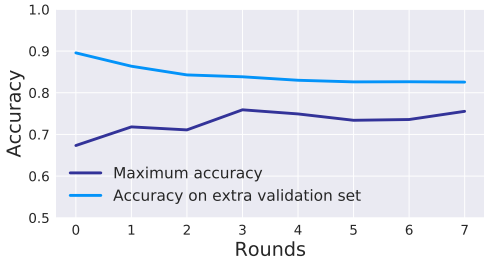
is still higher than 80%, which is enough to give confidence choosing those parameters.

- Mode 4: In this case, one can once more clearly observe the decrease in extra validation set accuracy, but only a slight change on classifier performance. Considering that the accuracy on the validation set is still higher than 80%, we opted for choosing 6 pre-training rounds, and selection threshold equals 8.

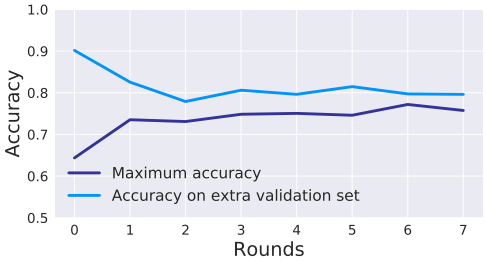


(a) Mode 1

(b) Mode 2



(c) Mode 3



(d) Mode 4

Figure 6.15: Performance of the selected models on each mode using the pre-training method on the extra validation set. Each mode has its peculiarities, but the accuracy decreases on the extra validation set, as the number of rounds increases, in all four modes.

Although using a higher number of rounds can sound expensive, computationally speaking, this burden is only reflected in the training step. In fact, the pre-training does not change the classifier structure; it only affects the input data that trains those models. So, the best models for the four classification modes are presented in Table 6.7.[2]

Table 6.7: Best profile for all classification modes in Platform C.

Mode	R	N	s	B	d	Rounds	τ	F_1	gACC	mACC	stdACC	mACCv
1	1	600	3	40	9	3	9	0.8732	0.8878	0.8807	0.0766	0.8709
2	2	900	4	100	9	5	9	0.7846	0.8169	0.8215	0.1102	0.9418
3	2	700	4	40	9	3	9	0.7448	0.7588	0.759	0.1696	0.8709
4	2	900	4	40	9	6	8	0.7903	0.7714	0.7719	0.1657	0.7972

[2]mACCv represents the accuracy on extra validation set.

6.5 Conclusions

In Platform C we tested the proposed methodology on a new fault, "Hydrate/clogging in service line", on an already known fault, "Hydrate in the production/injection line", on a detection scenario, and on a multiclass classification problem. The results obtained in this chapter are more significant, considering that there are more events on this platform compared to the other platforms, and, consequently, the system modeled with more data is more relevant.

When dealing with a new failure, the best model achieved good accuracy when compared to Platforms A ($88.0 \pm 10.0\%$) and B ($83.6 \pm 11.8\%$), showing that for 3 different types of fault it was possible to assemble a model with high accuracy.

Modeling an already seen fault delivered a model with accuracy $88.1 \pm 7.7\%$, which is almost the same obtained on Platform A.

The multiclass classification case reached $77.2 \pm 16.6\%$ which is a reasonable result, considering that we made no changes on the model. To better treat this scenario it is necessary to make same adaptations on the model. The next chapter will present some possible solutions for this.

Thus, Platform C showed the robustness of the proposed methodology, proving it to be a general and efficient way to handle hydrate related problems.

Chapter 7

Conclusions and Future Works

7.1 Concluding Remarks

At the beginning of this project we had as the primary goal the application of machine learning and data analytics knowledge in the oil and gas context. So, we started defining which area could be benefited by the project, taking into account that this area had to fill our requirement: there had to be enough data.

Then, we decided to put our efforts on solving problems related to hydrate and some other faults that occur in the same equipment affected by hydrate. We followed this path based on the analysis of each type of failure impact on the company losses during the past few years. At the end of this study, we came up with a set of seven potential faults of interest.

Three datasets were presented in this work, each one of them was composed by measurements of a set of sensors related to the wells from a specific offshore oil platform, recorded by the OSI Soft PI System. Due to each platform structure and operation, each database had its particularities, them being very different from one another. Dealing with real data requires a lot of work to prepare the data, especially when it comes from a highly dynamic system, such as offshore oil platforms. So, to handle the data of a given platform we proposed a methodology based on the following steps: determine which of the faults of interest prevail on the data; determine the tags that cover all events; determine the nature of each tag; clean the data.

The machine learning model proposed is composed of four blocks. First, we employed a classic feature extraction, based only on statistical features that are typically used in the literature. The second block transforms the data from the feature space to a suitable space, that is more informative and facilitates the task of the Modeling block, which encapsulates the algorithm, which, in this work, consists in a Random Forest classifier.

As the fourth block of our model, we proposed a technique, referred to as pre-training, to increase the classifier accuracy by filtering the data that feeds the model during the training phase. The data that would serve as input to the classifier now feeds this block, which trains a model (also a Random Forest classifier) using cross-validation. Then, we associated a score for each sample based on how many times it correctly classified during the cross-validation. If a sample score was higher than a fixed value, then this sample fed the classifier; otherwise it was removed from the training phase.

We presented the results for each offshore oil platform in a manner we could assess the pre-training impact. After applying our technique, we increased the accuracy by 10% in Platform A, and by 4.85%, 3.93%, 8.57%, 11.86% in the four modes in Platform C. Platform B had no improvements on the results using the pre-training technique.

In this dissertation, we believed to have deployed a robust methodology to handle any oil offshore platform data, that can be adjusted to model any fault, reaching over 80% of accuracy in most of the tested single class scenarios and over 75% in the multiclass scenarios.

7.2 Research Directions

As future works, the multiclass scenario should be expanded to better handle any number of faults. A possible way to achieve this is to ensemble many one-class classifiers, one for each possible class, and then fit a dynamic model that considers the outputs from every classifier.

As a possible direction to improve the model performance, we propose changing the Random Forest classifier for the Gradient Boosting (GBM) [48, 49]. The idea behind GBM has some similar aspects to the Random Forest, it also trains many weak learners, but it does so sequentially, not independently as in RF. This algorithm has been showing outstanding performance in the past few years. Moreover, the model could also be boosted by considering other features on the feature extraction block.

The pre-training presented promising performance, but it is a work in progress. The natural way to improve it is by eliminating fewer samples, but guaranteeing that it eliminates only contaminated samples.

Bibliography

[1] IRMANN-JACOBSEN, T. B. "Flow assurance – A system perspective". November 2011. Available at: <https://www.uio.no/studier/emner/matnat/math/MEK4450/h11/undervisningsmateriale/modul-5/MEK4450_FlowAssurance_pensum-2.pdf>.

[2] BOJARSKI, M., TESTA, D. D., DWORAKOWSKI, D., et al. "End to end learning for self-driving cars", *Computing Research Repository*, v. abs/1604.07316, April 2016. Available at: <http://arxiv.org/abs/1604.07316>.

[3] CAMPBELL, M., HOANE, A., HSIUNG HSU, F. "Deep blue", *Artificial Intelligence*, v. 134, n. 1, pp. 57–83, January 2002.

[4] SILVER, D., HUANG, A., MADDISON, C. J., et al. "Mastering the game of Go with deep neural networks and tree search", *Nature*, v. 529, n. 7587, pp. 484–489, January 2016.

[5] HOLZINGER, A. "Trends in interactive knowledge discovery for personalized medicine: Cognitive science meets Machine Learning", *The IEEE Intelligent Informatics Bulletin*, v. 15, n. 1, pp. 6–14, December 2014.

[6] SCHMIDHUBER, J. "Multi-column deep neural networks for image classification". In: *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3642–3649, June 2012.

[7] MARR, B. "How much data do we create every day? The mind-blowing stats everyone should read". May 2018. Available at: <Forbes.com>.

[8] BRUSTAD, S., LKEN, K., WAALMANN, J. "Hydrate prevention using MEG instead of MeOH: Impact of experience from major Norwegian developments on technology selection for injection and recovery of MEG", *Offshore Technology Conference*, May 2005.

[9] SLOAN, E. D. "A changing hydrate paradigm – from apprehension to avoidance to risk management", *Fluid Phase Equilibria*, v. 228-229, pp. 67–74, February 2005.

[10] ARTHUR, C. "Tech giants may be huge, but nothing matches big data". August 2013. Available at: <https://www.theguardian.com/technology/2013/aug/23/tech-giants-data>.

[11] PETROLEUM COMPANY, B. "BP statistical review of World Energy 2018", v. 67, pp. 44, June 2018. Available at: <https://www.bp.com/content/dam/bp/en/corporate/pdf/energy-economics/statistical-review/bp-stats-review-2018-full-report.pdf>.

[12] YIN, S., LI, X., GAO, H., et al. "Data-Based Techniques Focused on Modern Industry: An Overview", *IEEE Transactions on Industrial Electronics*, v. 62, n. 1, pp. 657–667, January 2015.

[13] JARDINE, A. K., LIN, D., BANJEVIC, D. "A review on machinery diagnostics and prognostics implementing condition-based maintenance", *Mechanical Systems and Signal Processing*, v. 20, n. 7, pp. 1483–1510, October 2006.

[14] WITTEN, I. H., FRANK, E., HALL, M. A. *Data mining: practical Machine Learning tools and techniques*. Morgan Kaufmann Series in Data Management Systems. 4 ed. San Francisco, Morgan Kaufmann, 2016.

[15] LIU, W., TANG, B., HAN, J., et al. "The structure healthy condition monitoring and fault diagnosis methods in wind turbines: A review", *Renewable and Sustainable Energy Reviews*, v. 44, pp. 466–472, April 2015.

[16] GRALL, A., BÉRENGUER, C., DIEULLE, L. "A condition-based maintenance policy for stochastically deteriorating systems", *Reliability Engineering and System Safety*, v. 76, n. 2, pp. 167–180, May 2002. ISSN: 0951-8320.

[17] YAM, R. C. M., TSE, P., LI, L., et al. "Intelligent Predictive Decision Support System for Condition-Based Maintenance", *The International Journal of Advanced Manufacturing Technology*, v. 17, n. 5, pp. 383–391, February 2001.

[18] WIDODO, A., YANG, B.-S. "Support vector machine in machine condition monitoring and fault diagnosis", *Mechanical Systems and Signal Processing*, v. 21, n. 6, pp. 2560–2574, August 2007.

[19] HELMY, T., FATAI, A., FAISAL, K. "Hybrid computational models for the characterization of oil and gas reservoirs", *Expert Systems with Applications*, v. 37, n. 7, pp. 5353–5363, July 2010.

[20] SMOLA, A. J., SCHÖLKOPF, B. "A tutorial on support vector regression", *Statistics and Computing*, v. 14, n. 3, pp. 199–222, August 2004.

[21] SLOAN, E. D. "Fundamental principles and applications of natural gas hydrates", *Nature*, v. 426, pp. 353–363, November 2003.

[22] HAMMERSCHMIDT, E. G. "Formation of gas hydrates in natural gas transmission lines", *Industrial and Engineering Chemistry*, v. 26, n. 8, pp. 851–855, August 1934.

[23] SLOAN, E., KOH, C. *Clathrate hydrates of natural Gases*. Chemical Industries. 3 ed. Boca Raton, CRC Press, 2007.

[24] CARROLL, J. *Natural gas hydrates: A guide for engineers*. 3 ed. Boston, Gulf Professional Publishing, 2014.

[25] KVAMME, B., KUZNETSOVA, T., AASOLDSEN, K. "Molecular dynamics simulations for selection of kinetic hydrate inhibitors", *Journal of Molecular Graphics and Modelling*, v. 23, n. 6, pp. 524–536, June 2005.

[26] RIBEIRO, C. P., LAGE, P. L. "Modelling of hydrate formation kinetics: State-of-the-art and future directions", *Chemical Engineering Science*, v. 63, n. 8, pp. 2007–2034, April 2008.

[27] TAVASOLI, H., FEYZI, F., DEHGHANI, M. R., et al. "Prediction of gas hydrate formation condition in the presence of thermodynamic inhibitors with the Elliott–Suresh–Donohue Equation of State", *Journal of Petroleum Science and Engineering*, v. 77, n. 1, pp. 93–103, April 2011.

[28] HOAGLIN, D. C., MOSTELLER, F., (EDITOR), J. W. T. *Understanding robust and exploratory data analysis*. 1 ed. New York, Wiley-Intersience, 2000.

[29] KOKOSKA, S., ZWILLINGER, D. *CRC standard probability and statistics tables and formulae*. 1 ed. Boca Raton, CRC Press, 1999.

[30] DECARLO, L. T. "On the meaning and use of kurtosis", *Psychological Methods*, v. 2, 1997.

[31] FENG, C., HONGYUE, W., LU, N., et al. "Log-transformation and its implications for data analysis", *Shanghai Archives of Psychiatry*, v. 26, n. 2, pp. 105–109, April 2014.

[32] BISHOP, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. 1 ed. New York, Springer-Verlag, 2006.

[33] AGGARWAL, C. C., HINNEBURG, A., KEIM, D. A. "On the surprising behavior of distance metrics in high dimensional spaces". In: *Proceedings of the 8th International Conference on Database Theory*, Lecture Notes in Computer Science, pp. 420–434, Berlin, 2001. Springer.

[34] BREIMAN, L. "Random Forests", *Machine Learning*, v. 45, n. 1, pp. 5–32, October 2001.

[35] DÍAZ-URIARTE, R., ALVAREZ DE ANDRÉS, S. "Gene selection and classification of microarray data using Random Forest", *BMC Bioinformatics*, v. 7, n. 1, pp. 3, January 2006.

[36] BELGIU, M., DRĂGUŢ, L. "Random forest in remote sensing: A review of applications and future directions", *ISPRS Journal of Photogrammetry and Remote Sensing*, v. 114, pp. 24–31, April 2016.

[37] JIA, J., LIU, Z., XIAO, X., et al. "pSuc-Lys: Predict lysine succinylation sites in proteins with PseAAC and ensemble random forest approach", *Journal of Theoretical Biology*, v. 394, pp. 223–230, April 2016.

[38] BREIMAN, L. "Bagging predictors", *Machine Learning*, v. 24, n. 2, pp. 123–140, August 1996.

[39] SEGAL, M. R. "Machine Learning benchmarks and Random Forest regression". April 2004. Available at: <`https://escholarship.org/uc/item/35x3v9t4`>.

[40] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., et al. "Scikit-learn: Machine Learning in Python", *Journal of Machine Learning Research*, v. 12, pp. 2825–2830, October 2011.

[41] OSHIRO, T. M., PEREZ, P. S., BARANAUSKAS, J. A. "How many trees in a Random Forest?" In: *Proceedings of the 8th International Conference on Machine Learning and Data Mining in Pattern Recognition*, MLDM 2012, pp. 154–168, Berlin, Heidelberg, July 2012.

[42] FRÉNAY, B., VERLEYSEN, M. "Classification in the presence of label noise: A survey", *IEEE Transactions on Neural Networks and Learning Systems*, v. 25, n. 5, pp. 845–869, May 2014.

[43] ABU-MOSTAFA, Y. S., MAGDON-ISMAIL, M., LIN, H.-T. *Learning From Data*. AMLBook, 2012.

[44] JAMES, G., WITTEN, D., HASTIE, T., et al. *An introduction to statistical learning: With applications in R*. 1 ed. New York, Springer-Verlag, 2013.

[45] BERGMEIR, C., HYNDMAN, R. J., KOO, B. "A note on the validity of cross-validation for evaluating autoregressive time series prediction", *Computational Statistics and Data Analysis*, v. 120, pp. 70–83, April 2018.

[46] SASAKI, Y. "The truth of the F-measure". October 2007. Technical Report – University of Manchester, School of Computer Science.

[47] TUKEY, J. W. "Exploratory Data Analysis". In: *The Concise Encyclopedia of Statistics*, 1 ed., Massachusetts, Addison-Wesley, 1977.

[48] CHEN, T., GUESTRIN, C. "XGBoost: a scalable tree boosting system". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, Sa Francisco, August 2016.

[49] FRIEDMAN, J. H. "Stochastic gradient boosting", *Computational Statistics and Data Analysis*, v. 38, n. 4, pp. 367–378, February 2002.