Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Nicola Tamascelli

A Machine Learning Approach to Predict Chattering Alarms

Master's thesis in RAMS Engineering
Supervisor: Nicola Paltrinieri
February 2020

NTNU
Norwegian University of Science and Technology

Nicola Tamascelli

A Machine Learning Approach to Predict Chattering Alarms

Master’s thesis in RAMS Engineering
Supervisor: Nicola Paltrinieri
February 2020

Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

NTNU
Norwegian University of
Science and Technology

Abstract

The alarm system plays a vital role to grant safety and reliability in the process industry. Ideally, an alarm should inform the operator about critical conditions only, and a set of corrective actions should be associated with each alarm. During alarm floods, the operator may be overwhelmed by several alarms in a short time span. Crucial alarms are more likely to be missed during these situations. Poor alarm management is one of the main causes of unintended plant shut down, incidents and near misses in the chemical industry. Most of the alarms triggered during a flood episode are nuisance alarms –i.e. alarms that do not communicate new information to the operator, or alarms that do not require an operator action. Chattering alarms –i.e. that repeat three or more times in a minute, and redundant alarms –i.e. duplicated alarms, are common forms of nuisance. Identifying nuisance alarms is a key step to improve the performance of the alarm system. Advanced techniques for alarm rationalization have been developed, proposing methods to quantify chattering, redundancy and correlation between alarms. Although very effective, these techniques produce static results. Machine learning appears to be an interesting opportunity to retrieve further knowledge and support these techniques. This knowledge can be used to produce more flexible and dynamic models, as well as to predict alarm behaviour during floods. The aim of this study is to develop a machine learning-based algorithm for real-time alarm classification and rationalization, whose results can be used to support the operator decision-making procedure. Specifically, efforts have been directed towards chattering prediction during alarm floods. Advanced techniques for chattering, redundancy and correlation assessment have been performed on a real industrial alarm database. A modified approach has been developed to dynamically assess chattering, and the results have been used to train three different machine learning models, whose performance has been evaluated and discussed.

Acknowledgements

First of all, I would like to express my sincere gratitude to Professor Nicola Paltrinieri of the Department of Mechanical and Industrial engineering at NTNU for his motivation and extensive knowledge. He supported me during all my stay at NTNU, providing crucial assistance and valuable insights for the development of my master thesis.

Further, I would like to thank my supervisor Sarah Bonvicini of DICAM at UNIBO for giving me the opportunity to write my master thesis in Norway and for the support during the drafting. Her enthusiasm and passion inspired me. Without her guidance, it would have been much more difficult to disentangle the formalities and the bureaucracy required to write the thesis abroad.

In addition, the assistance provided by Dr. Sandeep R. Kondaveeti, Professor Sirish L. Shah, Professor Valerio Cozzani and Dr. Tufan Arslan was greatly appreciated.

I would also like to acknowledge Yara Italia S.p.A for the data and the clarifications.

Furthermore, I must express my very profound gratitude to my family. They are a continuous source of inspiration and, without them, I would not be the person I am right now. There are no words enough to thank you for all you have done for me.

Finally, I would like to thank my friends for their constant presence in my life, for all the laughs and the wonderful moments. I am extremely lucky to have them in my life.

Table of contents

Chapter 1
Introduction

1.1. Background

The alarm system has always played a vital role to grant safety and reliability in the process industry. Before the advent of the DCS, the alarms were hard-wired (Katzel, 2007). Installing a new alarm was expensive (approx. 1000 $/alarm) (Katzel, 2007), and few alarms could be installed due to the limited space on the annunciator panel (Shaw, 1993). For these reasons, only crucial alarms were installed, and the need for a new alarm must have been carefully justified.

Nowadays, the alarm system is integrated with the DCS (Shaw, 1993; Katzel, 2007). Adding an alarm does not involve connecting cables and purchasing new hardware anymore (Shaw, 1993). Basically, installing new alarms has become "free". This has tremendously improved the flexibility of the alarm system, but some problems have arisen as well.

For instance, in modern industries, the ease of configuring new alarms has led to a large number of alarms being installed. Often, many of these alarms are added without proper rationalization (Kondaveeti *et al.*, 2010). As a result, the workload for the operator (i.e. the number of alarms to address) is often unbearable. Alarm floods and nuisance are problems that affect most of the modern chemical plants.

During alarm floods, the operator may be overwhelmed by hundreds of alarms in a short time span; in these situations, it is impossible to provide a timely response, and crucial alarms are more likely to be missed (Laberge *et al.*, 2014). Nuisance alarms do not communicate new information to the operator, or do not require an operator action (ANSI ISA, 2016). Chattering alarms –i.e. that repeat three or more times in a minute, and redundant alarms –i.e. duplicated alarms, are common forms of nuisance (Kondaveeti *et al.*, 2010). Typically, most of the alarms triggered during a flood episode are nuisance ones (Kondaveeti *et al.*, 2010; ANSI ISA, 2016). Identifying nuisance alarms is a key step to improve the performance of the alarm system.

Poor alarm management is one of the leading causes of unintended plant shut down, accidents, and near misses in the chemical industry (Stanton and Barber, 1995; Health and Safety Executive, 1997). Recently, standard manuals have been published (EEMUA, 2013; ANSI ISA, 2016), providing guidelines for effective alarm management and nuisance reduction. In addition, advanced alarm management techniques have been developed, proposing methods to quantify chattering, redundancy and correlation between alarms (Kondaveeti *et al.*, 2010, 2013; Yang *et al.*, 2012). But, although effective, these techniques produce static results. A chemical plant is not a static element, and so is the alarm system. In this "multivariate" context, the need of a dynamic and adaptive model is real.

We now live in the Digital Era; computational capabilities and data analysis techniques have extremely improved over the past few years. Industry 4.0, Digitalization and Internet of things (IoT) are deeply affecting the chemical industry (Ravi and Wu, 2016; Reis and Kenett, 2018). An immense amount of data can be stored in Cloud services and server farms. Still, extracting information and acquiring knowledge from raw data are not trivial tasks; unfortunately, data are stored but (often) not further analysed (Han, Kamber and Pei, 2012). Thus, the chance to acquire further knowledge from data is missed.

In this context, Machine Learning techniques have progressively captured the attention of the international scientific community (Liu *et al.*, 2018). These algorithms can "learn" from past data, and the knowledge achieved during the learning phase (i.e. training) can be used to predict future events (Brink, Richards and Fetherolf, 2016); hence, Machine Learning appears to be a good chance to use historical data to develop dynamic and flexible models.

1.2. Objective

The aim of this study is to develop a machine learning-based algorithm for real-time chattering prediction during alarm floods. In general, the method proposes an interesting opportunity to analyse historical alarm data and to extract knowledge from them.
The analysis includes the application of state-of-the-art techniques developed by Kondaveeti *et al.* (2010, 2013); this has been done to show the performances of the most recent alarm management techniques. From the results of these techniques, the work has proceeded through the development of a new, dynamic, method to assess chattering. Finally, the Machine Learning models have been developed and tested on their ability to predict chattering alarms.

The main objectives of this master's thesis can be summarized as follows:

1. the application of advanced alarm management techniques on a real industrial alarm database.
2. the development of a method to dynamically assess alarm chattering;
3. to use the results of the method mentioned above for training three different Machine Learning models: Linear, Deep and Wide&Deep;
4. to evaluate the capability of the models to predict alarm chatter.

1.3. Approach

A case study approach has been used in this thesis. All the analyses described in the present work have been performed on a real industrial alarm database, which was provided by the Norwegian chemical company Yara. *Figure 1.1* describes the analyses' workflow.
Firstly, the database has been studied and the main issues identified; time has been spent to become familiar with the database and with the plant layout. Secondly, the original database has been modified, and a new, more convenient, database has been created (Step 1 in *Figure 1.1*). Later, advanced alarm management techniques proposed by Kondaveeti *et al.* (2010, 2013) have been performed (Step 2).

Then, the original chattering index approach has been modified into a new, dynamic, method to assess alarm chatter (Step 3).

Finally, the results of the Dynamic chattering index method, along with alarm data from the original alarm database, have been used to train and evaluate three Machine Learning models (Step 4).

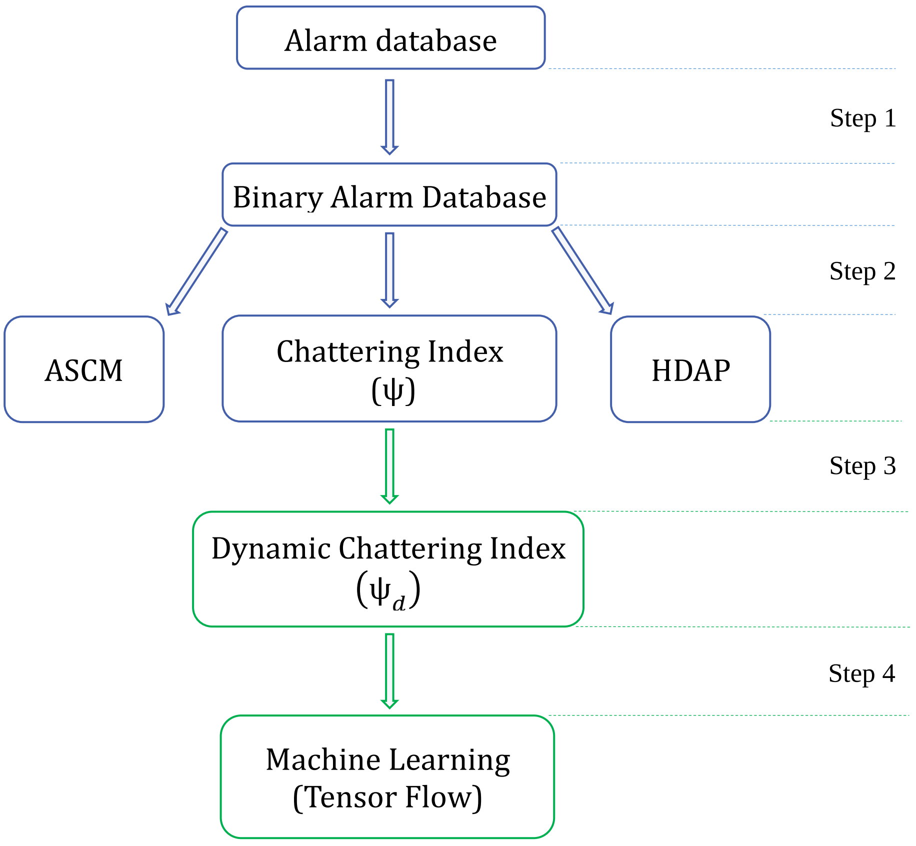


Figure 1.1 - Analyses workflow

In *Figure 1.1*, blue objects depict methods that have already been discussed in previous works (Kondaveeti *et al.*, 2010, 2013; Hu *et al.*, 2015). The green items represent original methods, developed during the present work.

The analyses have been performed using python as a programming language. PyCharm 2019.2 IDE has been used.

It is worth noting that the approach and the proposed method are limited to the case study under assessment. The results of the Machine Learning models are strictly related to the features of the plant under assessment (ammonia production, continuous operation, alarm flood episodes). Similarly, the method presented in this thesis has been developed with the sole purpose of predicting alarm chatter; using the same method to predict other metrics may not lead to the same results.

1.4. Outlines

This work includes seven chapters and three appendices. *Chapter 2* describes the theoretical background of the present work, and it comprises two sections. In the first section, the key concepts of “alarm” and “alarm system” are described, as well as state-of-the-art techniques for alarm management. In the second section, Machine Learning is introduced. *Chapter 3* focuses on the alarm database, which represents the case study of the present work. Furthermore, a brief description of the chemical plant associated with the alarm database is provided. In *Chapter 4*, the analyses performed during this thesis work are described in detail. Specifically, the first sections of the chapter focus on the application of the techniques proposed by Kondaveeti *et al.* (2010, 2013), and on the development of the Dynamic chattering index. The final section of the chapter focuses on the Machine Learning simulations. In *Chapter 5*, the results obtained from the analyses described in *Chapter* 4 are revealed. The results are discussed and evaluated in *Chapter 6*. Additionally, the limitations of the methods are highlighted and, finally, recommendations for further works are provided. In the final chapter (*Chapter 7*), the findings are summarized and framed into the context outlined in section *1.1*.

The three appendices include a list of acronyms (*Appendix A*), the code used for the Machine Learning simulations (*Appendix B*) and the tables that are either too large to be displayed in the main body or that are believed to be less relevant (*Appendix C*).

Chapter 2
Theoretical background

2.1. Introduction

In the next two subchapters, the key concepts about the alarm system and Machine Learning are presented.

In the first subchapter, the alarm system is described. First, the definitions of “alarm” and “alarm system” are provided, including their main features and related issues (e.g. nuisance). Secondly, the alarm management lifecycle is introduced, and how to properly manage and maintain the alarm system is described. Thirdly, an overview of the most significant metrics to evaluate the performance of the alarm system is provided. Unless otherwise specified, ANSI/ISA - 18.2 (2016) has been used as the main reference in these sections. Finally, state-of-the-art techniques for alarm management and rationalization are presented.

In the second subchapter, Machine Learning is introduced; including origins, development and actual applications. Next, the most important metrics to evaluate the performance of a Machine Learning classification algorithm are introduced. Finally, the models and the software used in this thesis are described.

2.2. The alarm system

According to the definition provided in ANSI/ISA - 18.2 (2016), the alarm system is a

collection of hardware and software that detects an alarm state, communicates the indication of that state to the operator, and records changes in the alarm state.

The alarm system represents a communication channel between the plant and the operator. During abnormal events, situations may arise where automatic systems (e.g. BPCS-Basic Process Control System, SIS-Safety Instrumented System) are not capable to restore normal process conditions; human intervention is needed to handle these situations. But, the first step to address a problem is being aware that a problem exists; through the alarm system, the operator is informed about abnormal process conditions or equipment malfunctions. The operator him/herself is part of the alarm system and can affect its performance. A well designed and reliable alarm system is an essential condition to grant a safe and stable plant.
A more detailed description of the alarm system is presented in *Figure 2.1*; arrows represent the dataflow between the elements of the system.

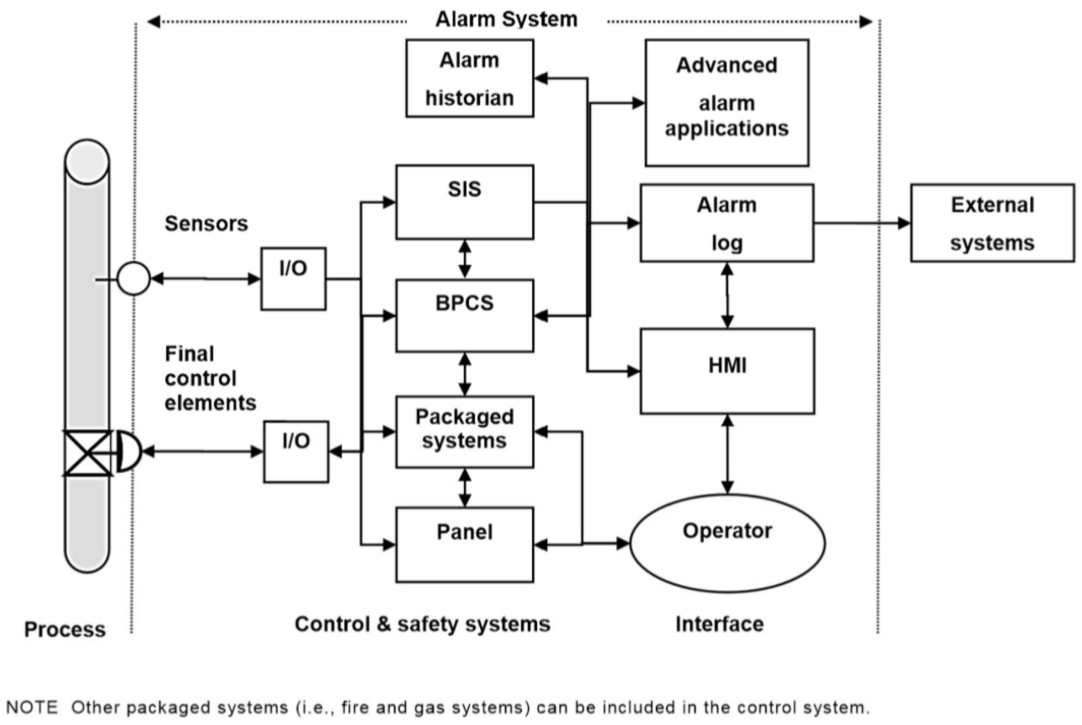


Figure 2.1 - Alarm system dataflow (ANSI ISA, 2016)

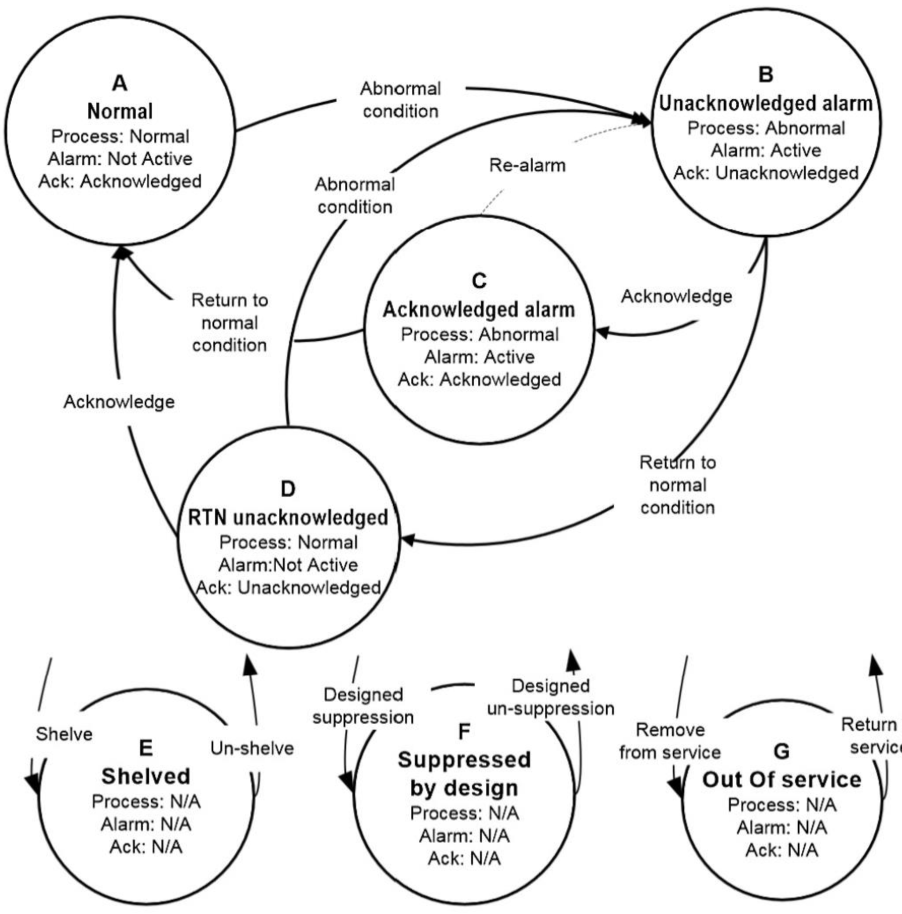
From the “Process”, data are sent to the “Control & safety system”, which comprises the Safety Instrumented System (SIS), the Basic Process Control System (BPCS), the “Packaged systems” and the “Panel”. Each element of the “Control & safety system” can communicate with the others. Then, data are sent to the “Interface” section, where alarm data are registered and stored (Alarm log and Alarm historian) and, finally, sent to the operator through the Human Machine Interface (HMI -e.g. a computer screen and a console). A two-way communication exists between the process and the operator, who does not passively receive information; the operator can affect the process conditions through the HMI, the panel and the packaged systems.
The alarm system is not a static element, it ages and degrades like all the other elements inside a plant. Thus, it needs to be managed and maintained to ensure good performances over time. Before going deeper into the description of the alarm system management, a fundamental element must be described: the alarm.

2.2.1. The alarm

According to the definition provided in ANSI/ISA - 18.2 (2016), an alarm is

> *an audible and/or visible means of indicating to the operator an equipment malfunction, process deviation, or abnormal condition requiring a timely response.*

It is worth noting that each alarm requires a *timely response*. If an alarm cannot be solved (i.e. no actions available or not enough time to respond), the alarm is ineffective and unnecessary. Typically, during an abnormal event, an alarm transitions into different *states*. The *state* of an alarm defines whether the alarm is active or not, as well as whether the operator has acknowledged the alarm. *Figure 2.2* depicts the possible transition paths for the majority of the alarms.



NOTE 1 States E, F, and G can connect to any alarm state in the diagram.
NOTE 2 The dotted line indicates an infrequently implemented option.
NOTE 3 N/A indicates not applicable or that the condition is not relevant in the alarm state.

Figure 2.2 - Alarm state transition diagram (ANSI ISA, 2016)

During normal operations, an alarm is *not active*, and its state is represented by the circle labelled as "A" in *Figure 2.2*. An abnormal event may occur, and the alarm state switches to "B" *active* and *unacknowledged* (because the operator response to an alarm is not instantaneous). Then, the alarm state may proceed along two different paths:

1. the alarm returns to *normal condition* without being acknowledged (B → D);
2. the alarm is acknowledged by the operator (B → C).

In case *1*, the control system (e.g. BPCS, SIS, etc.) solved the abnormal event without human intervention, and before the operator has acknowledged the problem; the alarm state is *not active* and *unacknowledged* (C). Then, when the operator acknowledges that the alarm has been solved, the alarm state returns to "A", or, if an abnormal event occurs again, the alarm state returns to "B".

In case *2*, the operator acknowledges the active alarm (typically pressing a button) before normal process conditions are restored; the alarm state is *active* and *acknowledged*. Then, the process may return to normal operation (A) or a new abnormal event may arise (B). A special case is the transition from “C” to “B”, this happens when an alarm has been acknowledged but the situation does not return normal in a reasonable time. In this situation, an alarm may be built to re-activate after a pre-defined amount of time.
Circles “E”, “F” and “G” represent special cases of alarm *states*:

- “Shelved”: temporarily suppressed by the operator;
- “Suppressed-by-design”: temporarily suppressed based on plant operating condition (i.e. start-up, maintenance, tests, etc.);
- “Out-of-service”: manually suppressed and removed from service (e.g. for maintenance).

As previously argued, timing is a key concept in managing alarms. A typical alarm response timeline is described in *Figure 2.3*.

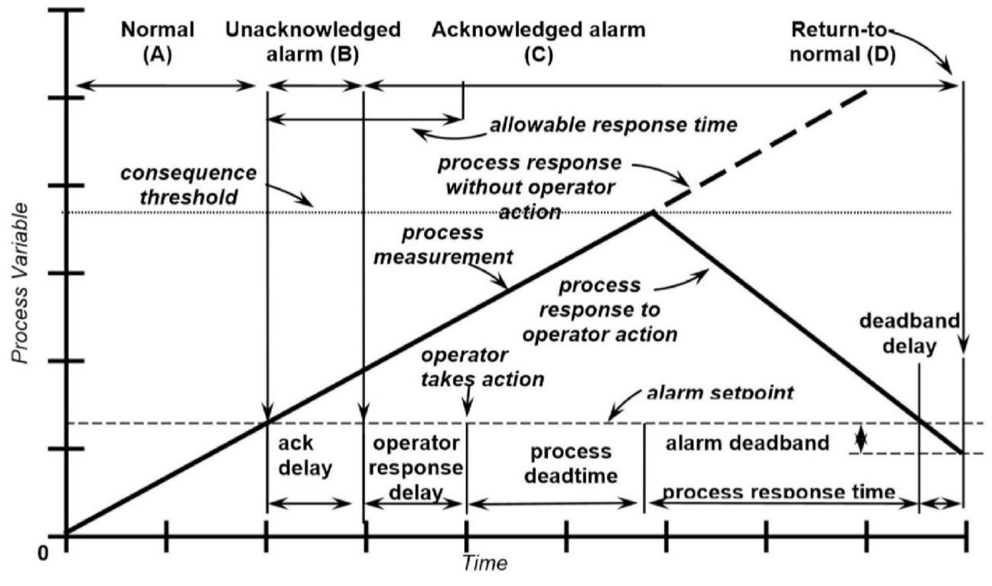


Figure 2.3 - Alarm response timeline (ANSI ISA, 2016)

The figure above represents a process value that increases over time (solid line); alarm *states* at different times (according to *Figure 2.2*) are described on the top of *Figure 2.3*. When the process value crosses the alarm setpoint (i.e. an alarm design attribute, see *2.2.1.2*), the alarm state turns to *active*. Then, after a certain amount of time, the operator acknowledges the alarm; the amount of time between the alarm and the acknowledgment is the *ack delay*. After the operator has acknowledged the alarm, he/she takes action to return to normal operations. The amount of time between the acknowledgement and the action is the *operator response delay*, which is a function of several factors, such as:

- operator workload;
- the complexity of determining the operator action;
- the complexity of the operator action;
- operator awareness and training;
- operator console clarity and ergonomics.

The sum of the *ack delay* and the *operator response delay* is the *actual response time,* which is bounded from above by the *allowable response* time. If an action is taken after the *allowable response time* the consequence will occur in any case. *Process deadtime*, rate of change of the process variable and the difference between the *consequence threshold* and the *alarm setpoint* are characteristics that influence the *allowable response time*. If the correct actions are taken in time, the process variable will start to decrease after the *process dead time*, eventually reaching the *alarm setpoint* again. Typically, the alarm does not *return-to-normal* immediately after crossing the *setpoint*, a *deadband delay* is set to prevent the alarm from turning on and off frequently if the process variable fluctuates around the *alarm setpoint*.
If the wrong actions are taken (or the correct actions are taken too late) the process variable continues to increase, and the consequences occur (dashed line in *Figure 2.3*).

2.2.1.1. Nuisance alarm

According to the definition provided in ANSI/ISA - 18.2 (2016), a nuisance alarm is:

an alarm that annunciates excessively, unnecessarily, or does not return to normal after the operator action is taken.

Basically, a nuisance alarm does not provide any new information to the operator, or there are no possible actions to solve the alarms (Kondaveeti *et al.*, 2010); thus, it constitutes a distraction for the operator. It is mandatory to periodically assess and reduce the number of nuisance alarms to grant a stable and efficient alarm system.

Examples of nuisance alarms are:

a. chattering alarms;
b. fleeting alarms;
c. stale alarms;
d. redundant alarms.

According to ANSI/ISA - 18.2 (2016), a chattering alarm

repeatedly transitions between the active state and the not active state in a short period of time.

Within a few hours, or even minutes, a chattering alarm could be triggered hundreds of times. Obviously, the operator has no chance to manage such a vast amount of alarms. A rule of thumb to determine chattering behaviour is 3 or more alarm records (from the same alarm) in one minute (Kondaveeti *et al.*, 2013).

Fleeting alarms share the characteristic of rapid transition between the active and not active state but, unlike chattering alarms, they do not do it repeatedly (i.e. with high frequency). Stale alarms are alarms that stay active for a long time (e.g. more than a day). Finally, redundant alarms are two or more alarms that always occur together (e.g. they are associated with the same process variable).

2.2.1.2. Alarm types and attributes

Different *types* of alarm exist in a plant, for instance:

a. *absolute alarm*: alarm generated when the alarm setpoint is exceeded (e.g. high-high, high, low, low-low);
b. *discrepancy alarm*: alarm generated by the difference between the expected plant or device state to its actual state (e.g., when a motor fails to start after it is commanded to the on state);
c. *calculated alarm*: alarm generated from a calculated value instead of a direct process measurement;
d. *instrument diagnostic alarm*: alarm to indicate a field device or signal fault;
e. *bad-measurement alarm*: alarm generated when the signal for a process measurement is outside the expected range;
f. *adaptive alarm*: alarm for which the setpoint is changed by an algorithm.

The definitions provided in the previous list are entirely drawn from ANSI/ISA - 18.2. Furthermore, each alarm is characterized by a series of *attributes*, which define the behaviour of the alarm within the control system. These *attributes* may vary depending on the specific alarm *type*, and they include:

a. alarm description;
b. alarm setpoint;
c. alarm priority;
d. alarm deadband;
e. on-delay or off-delay;
f. alarm group;
g. alarm message.

Each of these attributes is important, but some of them directly affect how the alarm behaves during an abnormal event; a brief description is needed to further describe these "special" attributes:

- alarm setpoint: a threshold value that, when crossed, causes the alarm to transition into the *active* state. The alarm setpoint greatly affects the alarm performance, since it directly determines the allowable response time (see *Figure 2.3*). The alarm setpoint determination must follow a clear and rational method, that must consider the consequence threshold, the complexity of the operator actions, the normal operating range, etc.;

- alarm priority: as the name suggests, this attribute determines the urgency of the alarm. It supports the operator to decide in which order the alarms should be addressed. Priority is not just a matter of severity of the consequences; allowable response time must be considered as well. Typically, three or four priority levels are used. The alarm priority determination must follow a clear and rational method and, ideally, most of the alarms should have low priority levels, while only a few of them should have higher priority levels;

- alarm deadband: *Figure 2.4* clarifies the function of the deadband. The solid line represents the process variable. When the value crosses the upper, horizontal, solid line ("High Limit" in *Figure 2.4* -i.e. the alarm setpoint) a notification is sent to the operator. Then, due to measurement noise, the process value crosses three more times the alarm setpoint value, but no notification is sent to the operator; this is because of the deadband (represented as the dashed, horizontal, line in *Figure 2.4* -i.e. "High – DB"). If the process value stays between the setpoint and the deadband, no notification is sent to the operator. An accurate deadband setting can significantly reduce the number of nuisance alarms;

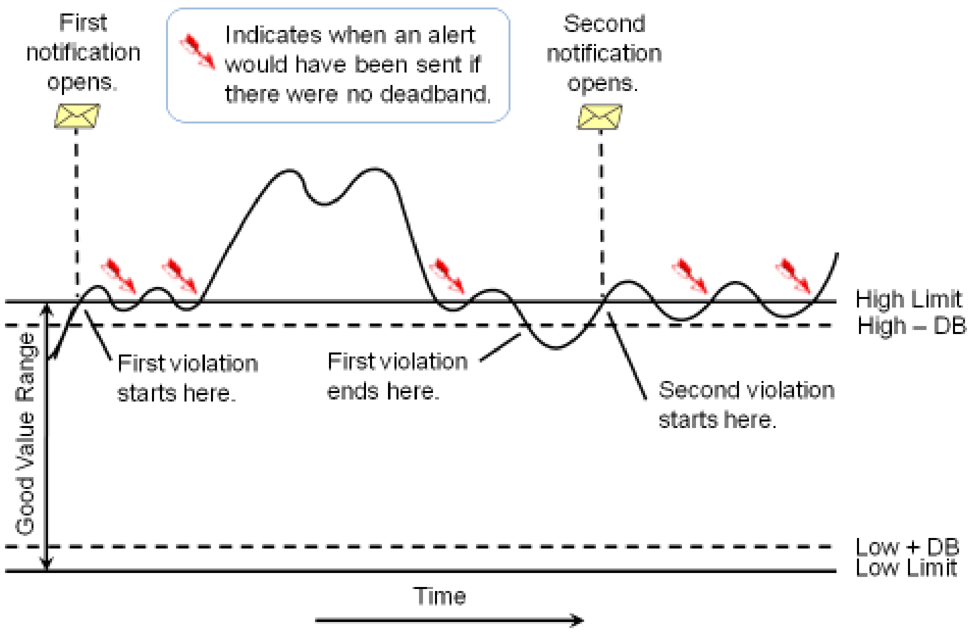

First notification opens.
Indicates when an alert would have been sent if there were no deadband.
Second notification opens.
High Limit
High – DB
First violation starts here.
First violation ends here.
Second violation starts here.
Good Value Range
Low + DB
Low Limit
Time

Figure 2.4 - Deadband and setpoint (livelibrary.osisoft.com, 2020)

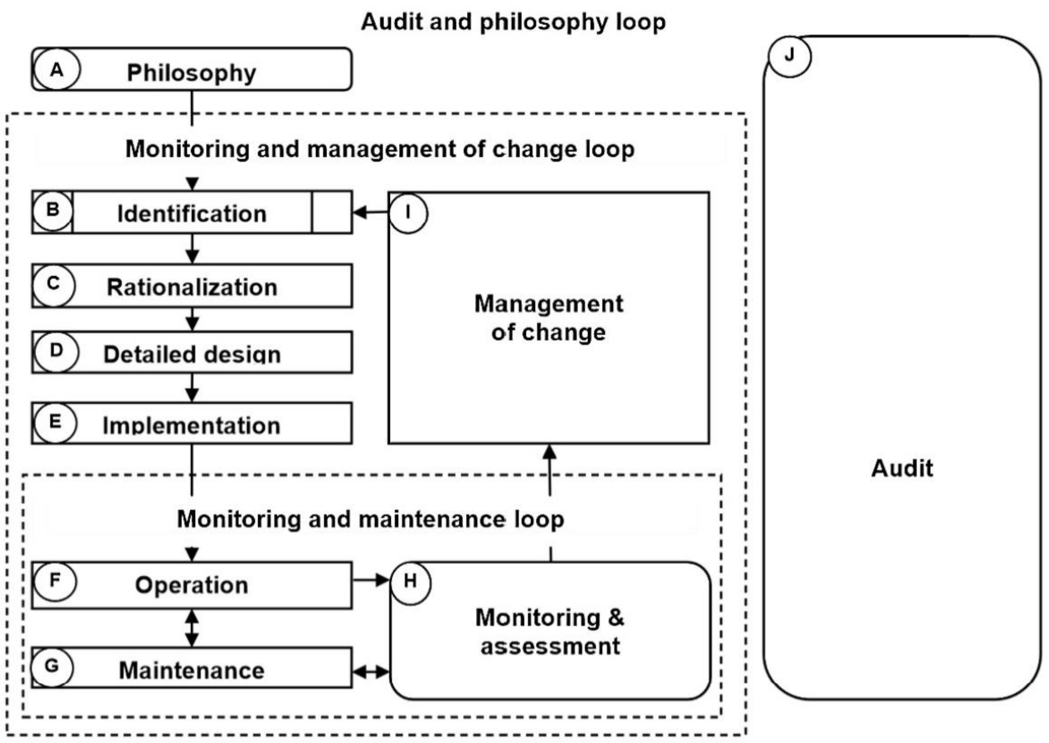
- <u>alarm off-delay</u>: a parameter that defines how long an alarm has to stay active after the process condition has returned normal. It is similar to the deadband, but it is based on a time value, instead of a process value. If an active alarm crosses the setpoint (and an eventual deadband) and no off-delay is set, the alarm turns *not active*. But, if an off-delay of one minute is set, the alarm stays active one minute more, no matter if the process condition has returned normal already. An accurate off-delay tuning can significantly reduce chattering.

Alarm attributes are decided during the basic design phase of the alarm system (*2.2.2* points *C* and *D*), and they are not static parameters. They can be "manually" changed to address a known nuisance problem, or they can be programmatically changed based on the current plant state (e.g. start-up, normal operation, etc.). Every time the

alarm attributes are changed, the operator must be informed about the change. Every change must be authorized and approved.
It is worth noting that alarm attributes can be changed also by advanced alarming techniques, which is the scope of this thesis work. An example is the “Model-based alarming” technique, according to which the alarms’ behaviour (e.g. attributes) can be changed based on a model prediction if a reliable model is available. For instance, the model could predict the plant state or the alarm behaviour, and it could change the alarms attributes to adapt the alarm system to the upcoming conditions.

2.2.2. The alarm management lifecycle

The alarm system needs to be properly managed and maintained to ensure its effectiveness. ANSI/ISA - 18.2 (2016) proposes a lifecycle-based alarm management, which comprises ten stages and three internal loops; a schematic description of the alarm management lifecycle is presented in *Figure 2.5*.
Either if a new alarm system is installed, or an existing one needs to be managed, the alarm management lifecycle will provide a rational method to ensure an efficient system.



NOTE 1 The box used for stage B represents a process defined outside of this standard per 5.2.2.3.
NOTE 2 The independent stage J represents a process that connects to all other stages per 5.2.2.11
NOTE 3 The rounded shapes of stages A, H, and J represent entry points to the lifecycle per 5.2.3.
NOTE 4 The dotted lines represent the loops in the lifecycle per 5.2.5.

Figure 2.5 - Alarm management lifecycle (ANSI ISA, 2016)

It is worth noting that, according to "note 3" in *Figure 2.5*, that one can enter the alarm lifecycle through the "Philosophy" stage (A), the "Monitoring & assessment" stage (H) or the "Audit" stage (J). A brief description of each stage will now be provided:

A. *Philosophy*

   the Philosophy stage constitutes the foundations of the whole alarm management lifecycle. During this stage, a document must be drawn, containing the criteria, definitions, principles and responsibilities of the alarm management lifecycle. The alarm Philosophy provides the method that must be followed by the other stages of the lifecycle to achieve their purposes. Recommended/required topics that must be covered over the alarm Philosophy are presented in *Table C. 3*. For example, the philosophy must clarify the purpose of the alarm system, the methods for the alarm design (i.e. how to calculate setpoints, deadbands, off-delay, alarm types, etc.), the basis and the metrics used for alarm prioritization, the methods for monitoring and maintaining the alarm system, and much more. Basically, it provides the guidelines on how to perform each stage of the alarm management lifecycle, and it constitutes the natural entry point for new systems.

B. *Identification*

during this stage, a collection of potential alarms is provided. The identification method (i.e. how to quantify the need for a new alarm) must follow the guideline presented in the Philosophy stage. An alarm may be identified by formal methods (such as HAZOP, FMEA, P&ID reviews, etc.) or by operational experience and plant knowledge. The output of the Identification stage (list of potential alarms) is the input to the Rationalization stage.

C. *Rationalization*

first, during the Rationalization stage, the need for each potential alarm must be justified (it must be ensured that the alarm meets the criteria of the alarm Philosophy). During the justification phase, it should be also verified that the potential alarm does not duplicate an existing alarm, and that it will not become a nuisance. If the alarm is consistent with the Philosophy, the alarm setpoint is determined as well as the alarm priority and classification. The list of partially determined alarms is then sent to the “Detailed design” stage.

D. *Detailed design*

during this stage, the alarm if fully designed and determined. Additional alarm attributes are specified (e.g. deadbands, off-delay, etc.), HMI is designed (e.g. how the alarm is presented to the operator based on the priority, the state, etc.) and advanced alarming is designed. The latter is used if the basic alarm design is not sufficient to grant the performances required by the alarm Philosophy. An example is the “Model-based alarming” technique, which was introduced at the end of paragraph *2.2.1*.

E. *Implementation*

during this stage, the alarms are physically installed and tested. Finally, the operators are trained.

F. *Operation*

the alarm/the alarm system is operative.

G. *Maintenance*

in this stage, the alarm is not operative because tests or reparation are needed. Periodical maintenance on the alarm system is essential to sustain its performance.

H. *Monitoring & assessment*

during this phase, the performances of the alarm system are monitored. Alarm data are analysed, and performance metrics are produced (see *2.2.3* for more details). If the effectiveness of the alarm system does not match the Philosophy requirements, maintenance or changes to the alarm system may be required (e.g. different alarm attributes, new alarms, advanced alarming techniques, reparation, etc.). This is the natural entry point for existing alarm systems. Furthermore, the "Monitoring & assessment" stage is the entry point for the techniques discussed in this work; since the aim of this thesis is to provide a method to address nuisance and enhance the alarm system performances. The output of this stage is a list of suggestions to improve the performances.

I. *Management of change*

in this stage, the changes identified during "Monitoring & assessment" are discussed and approved. The output of this stage is a list of authorized changes, which is fed to the "Identification" stage.

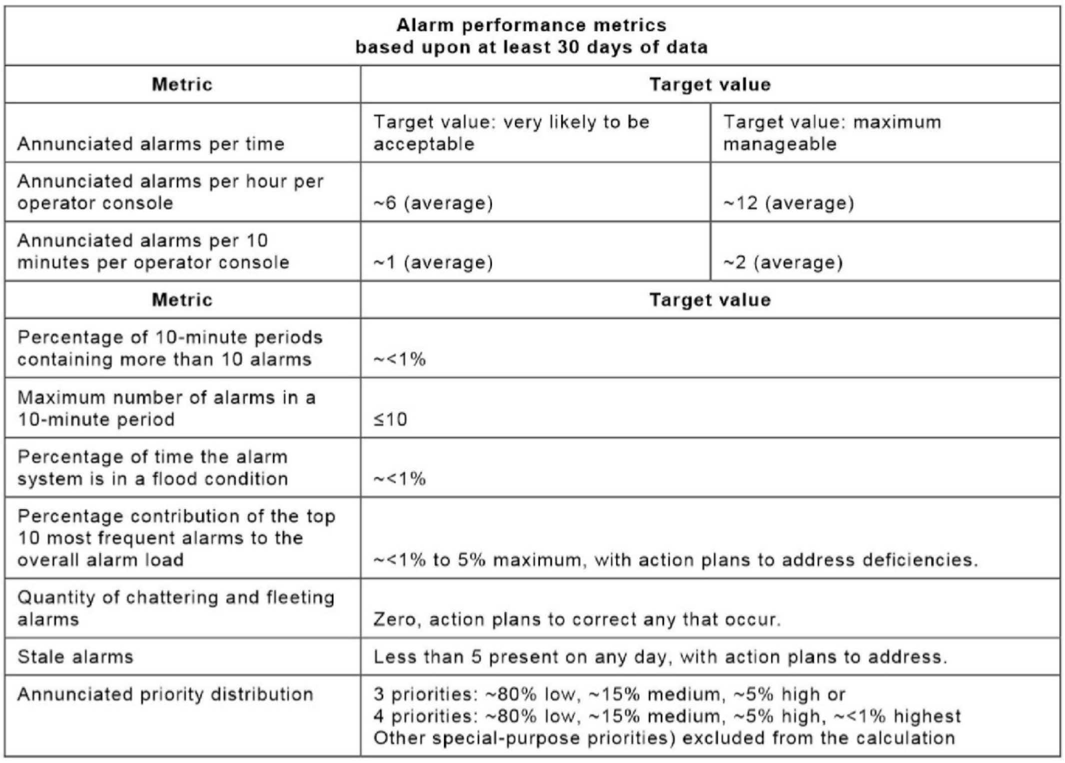
J. *Audit*

this is a separate stage of the alarm management lifecycle. It is periodically conducted to preserve the efficiency of the alarm system and the alarm management lifecycle itself. This is the only phase where modifications to the Philosophy can be discussed and, eventually, approved. Audit stage may highlight issues not recognizable by the "Monitoring & assessment" stage.

In *Table C. 4*, a concise description of the activities performed in each stage of the alarm management lifecycle is provided, along with the inputs and the outputs.

2.2.3. Performance of the Alarm system

During the “Monitoring & assessment” stage, the performance of the alarm system must be monitored and evaluated against the Philosophy requirements. Various performance metrics exist to assess the alarm system's effectiveness. All the metrics are calculated from alarm data (i.e. an alarm database, a collection of alarm records) and, usually, at least thirty days of alarm data are needed. The metrics suggested by ANSI/ISA 18.2 are summarized in *Table 2.1*.



Alarm performance metrics based upon at least 30 days of data		
Metric	**Target value**	
Annunciated alarms per time	Target value: very likely to be acceptable	Target value: maximum manageable
Annunciated alarms per hour per operator console	~6 (average)	~12 (average)
Annunciated alarms per 10 minutes per operator console	~1 (average)	~2 (average)
Metric	**Target value**	
Percentage of 10-minute periods containing more than 10 alarms	~<1%	
Maximum number of alarms in a 10-minute period	≤10	
Percentage of time the alarm system is in a flood condition	~<1%	
Percentage contribution of the top 10 most frequent alarms to the overall alarm load	~<1% to 5% maximum, with action plans to address deficiencies.	
Quantity of chattering and fleeting alarms	Zero, action plans to correct any that occur.	
Stale alarms	Less than 5 present on any day, with action plans to address.	
Annunciated priority distribution	3 priorities: ~80% low, ~15% medium, ~5% high or 4 priorities: ~80% low, ~15% medium, ~5% high, ~<1% highest Other special-purpose priorities) excluded from the calculation	

Table 2.1 – Recommended alarm performance metrics summary (ANSI ISA, 2016)

A brief description of the metrics is presented below.

1. *Average alarm rate per operator console*

   number of annunciated alarms per operator based upon one month of data (i.e. thirty-day average). The following limits are suggested:
   - acceptable: ~ 6 alarms per hour per operator (average);
   - maximum: ~ 2 alarms per ten minutes per operator (average).

   The thresholds presented above consider the experience of the operator and the time needed to study the situation, to take corrective actions and to verify that the situation has returned normal.

2. *Peak alarm rate per operator console*

   an operator cannot handle more than 10 alarms in a 10-minutes interval. Peak alarm rate analysis consists in dividing the month into 10-minutes-spaced intervals. For each interval, the number of annunciated alarms per operator is calculated. The number of intervals containing more than ten alarms represents the "Peak alarm rate per operator console". The recommended value is less than 1% (i.e. less than 43.2 ten-minutes intervals in a month). "Peak alarm rate per operator console" and "Average alarm rate per operator console" must be considered simultaneously.

3. *Alarm floods*

   Alarm floods are periods of intense alarm activity. Hundreds (or even thousands) of alarms may occur during a flood episode; in situations like this, crucial alarms are more likely to be missed. The duration of an alarm flood is variable; it starts when the alarm rate exceeds 10 alarms/operator per ten minutes time interval, and it ends when the alarm rate returns normal (e.g. less than 5 alarms/operator per ten minutes time interval). It is recommended that the alarm system should not experience floods for more than 1 % of the total time.

4. *Frequently occurring alarms*

   usually, in a chemical plant, hundreds of alarms are configured. However, only a few of them are responsible for most of the total alarms count (i.e. from ten, up to twenty alarms only are responsible for more than 70% of the total alarm occurrences within the study period). Addressing these frequent alarms can greatly enhance the alarm system performance. As a recommendation, the top 10 most frequent alarms (namely, 'top 10 bad actors') should not constitute more than 5% of the total alarm occurrences.

5. *Chattering and fleeting alarms*

   chattering and fleeting alarms have already been defined in *2.2.1.1*. Chattering alarms are usually in the list of the "Frequently occurring alarms". Chattering and fleeting alarms are not tolerated in any way. If chattering or fleeting alarms are identified, actions must be taken to correct them.

6. *Stale alarms*
   stale alarms have already been defined in *2.2.1.1*. There is no long-term acceptance for these kinds of alarms, but it is tolerable to have less than five stale alarms per day.

7. *Annunciated alarm priority distribution*
   as it was already mentioned in *2.2.1.2*, alarms with higher priority should be annunciated less frequently compared to the ones with lower priority. The "Alarm priority distribution" quantifies the consistency of the alarm prioritization procedure.

2.2.4. Chattering, Redundancy and Correlation assessment

As previously argued, a key step to improve the performance of the alarm system is to remove nuisance alarms (*2.2.1.1* and *2.2.3*) and to address frequently occurring alarms (*2.2.3*). During the past years, advanced alarm management tools have been developed to quantify chattering (Kondaveeti *et al.*, 2010, 2013), redundancy and correlation (Kondaveeti *et al.*, 2010; Yang *et al.*, 2012; Ahmed *et al.*, 2013). These techniques represent the foundations of this thesis work; a brief description of each of them is presented in the next three paragraphs.

2.2.4.1. Chattering assessment: the chattering index (ψ)

In section *2.2.1.1* a rule of thumb to identify a chattering alarm is defined as 3 or more alarms in a minute. But the definition is vague, and no standard or guideline exists to quantify the chattering behaviour of an alarm. Kondaveeti *et al* (2013) proposed a method based on run length distributions to quantify alarm chattering. The method follows 5 steps:

1. binary alarm database creation;
2. run length (r) calculation;
3. Run Length Distribution (RLD) calculation;
4. Discrete Probability Function (DPF) calculation;
5. chattering index (ψ) calculation.

Each step will be described in detail in paragraphs *4.2* and *4.5*.
The result of the procedure is a chattering index ψ for each unique alarm that occurred within the study period. The chattering index ψ of an alarm can be interpreted as the "*mean frequency of annunciation of that alarm assuming that the abnormal event prevails for an indefinite period of time*" (Kondaveeti *et al.*, 2013), and it has the following properties:

- $\psi \in [0,1]$ (the closer to 1, the more the alarm shows chattering behaviour);
- ψ units are alarms/s.

A suggested rule to determine whether an alarm shows chattering behaviour is:

$$\psi > 0.05 \, \frac{alarms}{s} \tag{2.1}$$

This is because 0.05 alarm/s is equal to 3 alarms/min, which is the suggested value already discussed in *2.2.1.1*.

2.2.4.2. Correlation and redundancy assessment: the ASCM

Redundancy has already been discussed in *2.2.1.1*, while "correlation" must be described further. The "correlation" is a measure that indicates "how much" two alarms are similar. If two alarms are correlated, they tend to be annunciated together. This does not necessarily mean that two correlated alarms appear always at the same time; for example, one of them may occur two minutes after the other. But, if the same delay between two alarms happens frequently, it means that the two alarms are somehow correlated. For example, the first alarm could be a high-temperature alarm of a gas-phase batch reactor, while the second one a high-pressure alarm of the same reactor; they are not the *same* alarm (i.e. they are not redundant) but they are certainly correlated. In this example, the operator actions should be aimed at decreasing the temperature, rather than decreasing the pressure; solving the high temperature will solve the high pressure as well. In this example, it is trivial to recognize correlation; instead, in more complex systems, it may not be intuitive. Obviously, correlation is not a form of nuisance, it is a measure to quantify the relationship between alarms. It could be used to support the operator actions and to assess redundancy (i.e. if two alarms are "extremely" correlated, they probably will be redundant).

A method to assess the correlation between two alarms was proposed by Kondaveeti et al. (2010). It is based on the binary representation of alarm data and the application of the Jaccard measure, which measures the "distance" (i.e. the correlation) between two binary sequences (Lesot, Rifqi and Benhadda, 2009). The method develops through five steps:

1. binary alarm database creation;
2. padding each binary sequence with extra 1's;
3. calculation of similarity measure;
4. re-ordering of the similarity matrix;
5. colour coding.

Each step will be described in detail in paragraphs *4.2* and 4.4.
The result of the procedure is the Alarm Similarity Color Matrix (ASCM, *Figure 2.6*).

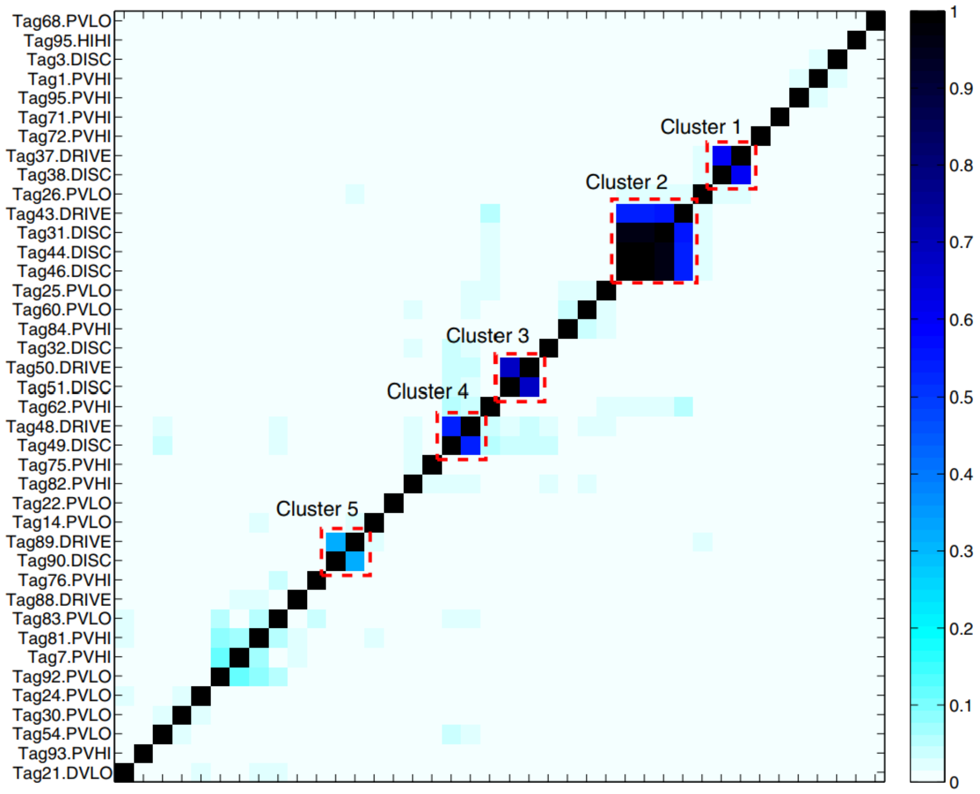


Figure 2.6 – An example of ASCM (Hu *et al.*, 2015)

Figure 2.6 introduces an example of an Alarm Similarity Color Matrix (ASCM), which is a symmetric matrix whose elements represent the degree of correlation between couples of alarms. The rows and the columns of the matrix represent a unique alarm. Each element of the matrix is displayed as a coloured square, the colour represents the value of the similarity measure (i.e. the Jaccard measure) between two alarms (the row and column index of the element). The Jaccard measure is bounded between 0 and 1; the higher the similarity (i.e. the correlation) the higher the Jaccard measure. Since the matrix is symmetric, the diagonal elements represent the correlation between one alarm and the alarm itself (i.e. the diagonal element of the row "Tag21.DVLO" in *Figure 2.6* represents the correlation between the binary sequences of "Tag21.DVLO" and "Tag21.DVLO"). But, the degree of correlation of two identical alarm is 1 (the binary sequence are identical); thus, every element in the diagonal has a Jaccard measure equal to 1 (maximum degree of correlation) and is represented as a black square, according to the colour bar on the right of *Figure 2.6*. Intuitively, the darker the colour of the matrix element, the higher the correlation between the two alarms. It is worth noting that the alarms are not randomly displayed in the matrix. Alarms are reordered (step *4* of the method) in such a way that alarms with higher correlation are displayed together in the matrix (Kondaveeti *et al.*, 2010), forming clusters of correlated alarms. If an alarm of a cluster is triggered, it is very likely that another alarm of the same cluster will be triggered anytime soon. This information could be used to support the operator decision-making procedure. Furthermore, the ASCM is used to assess redundancy; for example, if two different alarms have a similarity measure close to 1, it is highly probable that they are redundant alarms. One of the two redundant alarms can be silenced since it does not provide any new information to the operator. To conclude, ASMC is not just a graphical tool, the coloured squares represent a "real" similarity value that, as discussed above, is a meaningful and valuable piece of information.

2.2.4.3. The High Density Alarm Plot (HDAP)

In (Kondaveeti *et al.*, 2010) the authors proposed another alarm visualization tool, the High Density Alarm Plot (HDAP), which can be used to support the findings obtained by the techniques discussed above (Chattering index and ASCM). The HDAP is a convenient way to display large alarm databases and can be used to visually recognize periods of plant instability as well as to preliminary assess chattering and redundancy.

To obtain the HDAP the following steps must be followed:

1. binary alarm database creation;
2. time bins creation and alarm count;
3. HDAP creation.

Each step will be described in detail in paragraph *4.2* and *4.3*

The result of the procedure is the HDAP (*Figure 2.7*).

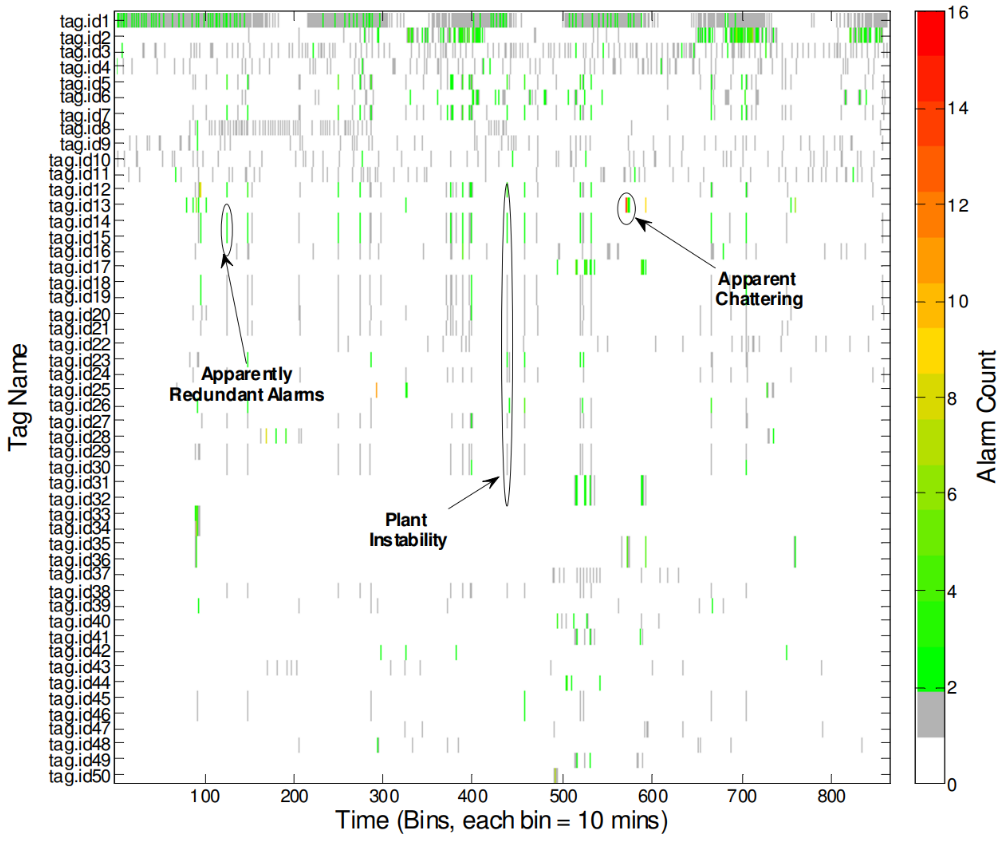


Figure 2.7 – An example of HDAP (Kondaveeti *et al.*, 2010)

Each row (i.e. point of the y-axis) represents the "*temporal representation of a unique alarm over the selected time range*" (Kondaveeti *et al.*, 2010), each column (i.e. point of the x-axis) represents a 10 minutes time interval (bin). The coloured sticks in the plot represent how many times the alarm of concern (row) is occurred within the time bin (column) according to the colour bar on the right of *Figure 2.7*. It is worth noting that the alarms are sorted in such a way that the total alarm count decreases from the top to the bottom of the plot (i.e. the first alarm of *Figure 2.7* -i.e. "tag.id1" has a higher total alarm count than the second one -i.e. "tag.id2", etc.). In this way, alarms with higher alarm count (i.e. the "Frequently occurring alarms", the "bad actors") are displayed on the top of the HDAP. Furthermore, redundant alarms tend to be displayed together since they have similar alarm count. The annotations in *Figure 2.7* clarify the usefulness of the HDAP; for instance, it can be used to recognize periods of plant instability, it can be used for preliminary redundancy assessment (alarms that appears always together in the plot, and with the same alarm count) and, finally, it can be used for preliminary chattering assessment (alarm with very high count within the time interval). It should be emphasized that the HDAP is just a visual tool, it is useful for a preliminary assessment, but it cannot substitute the two techniques described earlier (ψ and ASCM).

2.3. Machine Learning

Machine Learning is the field of Artificial Intelligence (AI) (Brink, Richards and Fetherolf, 2016) that comprises all the techniques (i.e. algorithms) through which a machine can gain knowledge from the past (i.e. past data), and use the acquired knowledge to perform several tasks (e.g. predictions, classification, pattern recognition, etc.).

There is not one, universally accepted, Machine Learning definition. In Mohri *et al.* (2012) it is defined as:

computational methods using experience to improve performance or to make accurate predictions.

The term "Machine Learning" was coined by Arthur L. Samuel (1959). He developed a computer algorithm to play checkers in such a way that the program "*will learn to play a better game of checker that can be played by the person who wrote the program*" (Samuel, 1959). The program was trained on playing thousands of games against itself; depending on the situation, the program learned the best moves (i.e. the moves that led to a victory). By 1970 the software achieved the level of an amateur player (Brink, Richards and Fetherolf, 2016), and this led to the birth of Machine Learning. Since then, Machine Learning techniques have progressively captured the attention of the international scientific community, and now they represent one of the "hot topics" of the 21st century (Liu *et al.*, 2018).

The actual applications of Machine learning are countless (Mohri, Rostamizadeh and Talwalkar, 2012; Brink, Richards and Fetherolf, 2016), the list below is just a quick and non-comprehensive review of the variety of different scientific fields that have taken advantage of Machine Learning techniques:

- computer vision tasks, e.g., image recognition, face detection;
- medical diagnosis;
- computational biology applications, e.g., protein function or structured prediction;
- text or document classification, e.g., spam detection;
- stock-market prediction;
- risk management.

And new applications are found every day.
Although dozens of Machine Learning algorithms exist, each of them falls into three big classes;

1. supervised learning;
2. unsupervised learning;
3. reinforcement learning.

In this thesis, only supervised learning has been used. Furthermore, supervised learning can be divided into two main categories: Regression and Classification. Since the aim of the present work is to classify alarms (i.e. the alarm “will show chattering” or “will not sow chattering”), Classification only has been used. Thus, in the following sections, the key concept about Classification problems, and related machine learning algorithms, will be provided (e.g. definitions, characteristics, performance metrics, tasks, models). In the final section, the software used during the simulations (TensorFlow) will be introduced.

2.3.1. Definitions and general aspects

Two definitions are needed before proceeding further into the description of unsupervised learning:

- *features*
  the features are meaningful attributes of the problem under assessment. The features should capture the relevant aspect of the problem (Brink, Richards and Fetherolf, 2016) and constitute the inputs of the Machine Learning model. For instance, if the task is to classify emails to detect spam, some relevant features may be the sender, the subject, the presence of specific keywords, etc. In this way, an email is completely described by a series of attributes. If the task is to predict alarms behaviour (like in this thesis), the features may include the alarm tag, the alarm status, the alarm attributes, the value of the associated process variable, etc.

- *labels* (or targets)
  the labels are the values or categories that the model has to predict. For instance, in the spam detection example, the label associated with an email is "Spam" or "Not Spam". In this thesis work, since the objective is to predict chattering, the labels may be "The alarm is going to show chattering" or "The alarm is not going to show chattering".

Supervised learning develops through two main steps:

1. training;
2. evaluation.

First, the original dataset (i.e. a database of features and associated labels) is divided into two distinct datasets (e.g. in a half):

a. the training dataset;
b. the evaluation dataset.

During the training phase, the algorithm has access to the training dataset only, which contains both the features and the labels. The scope of the training step is to build a function f such that:

$$Y = f(X) + \varepsilon \qquad 2.2$$

where:

- $Y = labels;$
- $X = features\ (tipically\ a\ matrix);$
- $\varepsilon = noise.$

Thus, the aim of the training is to find a relationship (f) between the labels (Y) and the features (X) ignoring the data noise (ε) (Brink, Richards and Fetherolf, 2016). How the best function is found is out of the scope of this work; usually, the loss is minimized, for more see Brink *et al.* (2016) and Mohri *et al.* (2012). Hopefully, at the end of the training phase, a function that well represents the relationship between features and labels is found.

After the training phase, the performance of the algorithm needs to be tested. The knowledge gained during the training is now used to predict the labels of a new set of features; this is the evaluation phase. First, the labels are removed from the evaluation

dataset (remark: the algorithm has not come into contact with the evaluation dataset so far) and the unlabelled dataset is fed to the trained algorithm. The task here is to predict the labels of the new features. If the training was successful, the algorithm would be able to predict most of the new labels (i.e. the predictions would match the real labels).
A clarification about the nature of the predictions is needed. The output of a model is not a label itself, but a list of label's probabilities. For instance, in the emails example, the raw output of the algorithm is not simply "Spam", but a probability vector like [0.78, 0.22], where 0.78 is the probability of the label being "Spam" and 0.22 the probability of the label being "Not Spam". Then, comparing the probabilities with a threshold value, the raw output is converted into the label, that is finally returned by the program. By default, the probability threshold level is 0.5 (i.e. a certain label will be predicted if its probability is greater than 0.5). The threshold value can affect the performance of the algorithm (Google, 2020b).

In the next sections, the metrics used to quantify the performance of a machine learning algorithm will be described.

2.3.2. Performance of machine learning algorithms

Several metrics are used to quantify the performance of a Machine Learning algorithm. It is worth noting that the performance is strictly related to the evaluation phase; a model cannot be assessed based on the results of the training phase only. Before introducing the performance metrics, the definitions of True Positive, True Negative, False Negative and False Positive are needed.
In the emails example, one can represent the labels (i.e. "Spam", "Not Spam") as a binary sequence, where "1" is the label "Spam" and "0" is the label "Not Spam". With this notation:

- *True positive* (TP)
  a "True positive" occurs when the model correctly predicts the label "1" (i.e. during the evaluation phase, for one set of features, the model predicted the label to be 1, and the true label was 1 as well).

- *true negative* (TN)
  the model predicted the label to be 0, while the true label was 0.
- *false positive (FP)*
  the model predicted the label to be 1, while the true label was 0.
- *false negative (FN)*
  the model predicted the label to be 0, while the true label was 1.

Typically, a confusion matrix is used to display TP, FP and FN. An example of a confusion matrix is presented in *Figure 2.8*.

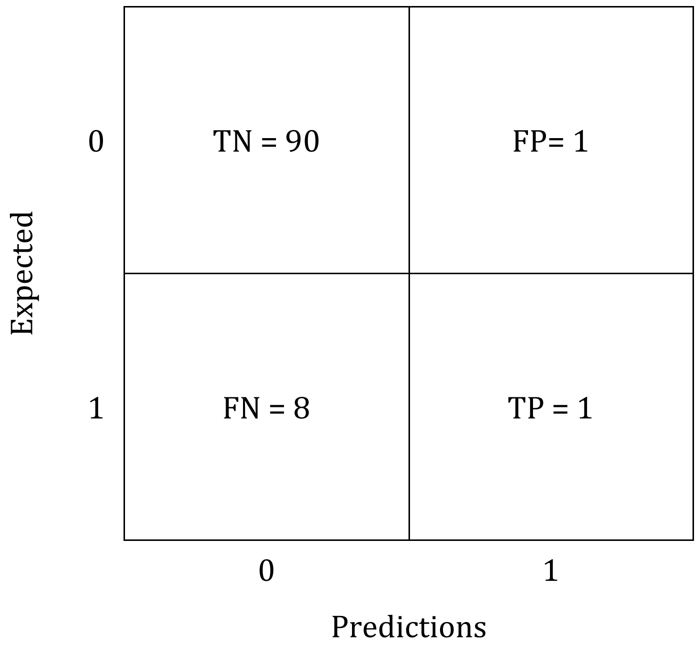


Figure 2.8 – The confusion matrix

The x-axis of *Figure 2.8* represents the predictions of the model (i.e. 0 and 1 – “Spam” and “Not Spam”) while the y-axis represents the real value of the labels. Looking at this matrix one can conclude that:

- the class “1” has been correctly predicted 1 time (TP);
- the class “0” has been correctly predicted 90 times (TN);
- the class “1” has been incorrectly predicted 1 time (FP);
- the class “0” has been incorrectly predicted 8 times (FN).

The confusion matrix is a useful tool to have a quick overview of the model performance, but it is not enough; performance needs to be further quantified.

Three metrics are widely used to assess the algorithm performance: Accuracy, Precision and Recall.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \qquad 2.3$$

$$Precision = \frac{TP}{TP + FP} \qquad 2.4$$

$$Recall = \frac{TP}{TP + FN} \qquad 2.5$$

Accuracy is the ratio between the correct predictions and the total number of predictions. Thus, it is a good starting point to evaluate the performance of the algorithm, but it is not enough. For instance, imagine that a Machine Learning algorithm to classify tumours have been created, the two labels to be predicted are "Benign" and "Malignant". The algorithm is evaluated on a dataset containing 100 tumours, 91 are benign and 9 are malign. Now, imagine that the model produced the results in *Figure 2.8*. Thus, the accuracy would be 0.91 (91 correct predictions out of a total of 100); it seems good. But a closer look at the results reveals that the model performance is totally unacceptable. In fact, of the 9 malign tumours, only 1 has been correctly predicted. This example (Google, 2020a) clarifies that accuracy alone is not enough, especially for unbalanced problems. Both precision and recall must be considered together.

The *Precision* is the fraction of correct positive predictions. The *Recall* is the fraction of real positive correctly predicted. In the tumour's classification example, according to the values in *Figure 2.8*, the precision would be 0.5 and the recall would be 0.11. The recall reveals that only 11 % of the actual malignant tumour have been correctly predicted; this is obviously not adequate.

All the metrics described above must be considered together but, depending on the problem of concern (e.g. spam identification, tumour identification, etc.), one metric is usually more significant than the others. For example, in the tumour classification problem, the Recall is the most important metric, because it is crucial to identify most of the malignant tumour. In the email classification, it is crucial to not classify legit emails as "Spam" ones; thus, precision is the metric that must be optimized.

Precision and Recall are affected by the threshold (i.e. the probability level beyond which the predicted label is "1") but, unfortunately, precision and recall are often "in tension" (Google, 2020b); usually, trying to improve one metric will cause the other to worsen. The Precision-Recall curve is a visualization tool that displays the precision and recall values varying the threshold. An example is presented in *Figure 2.9*.

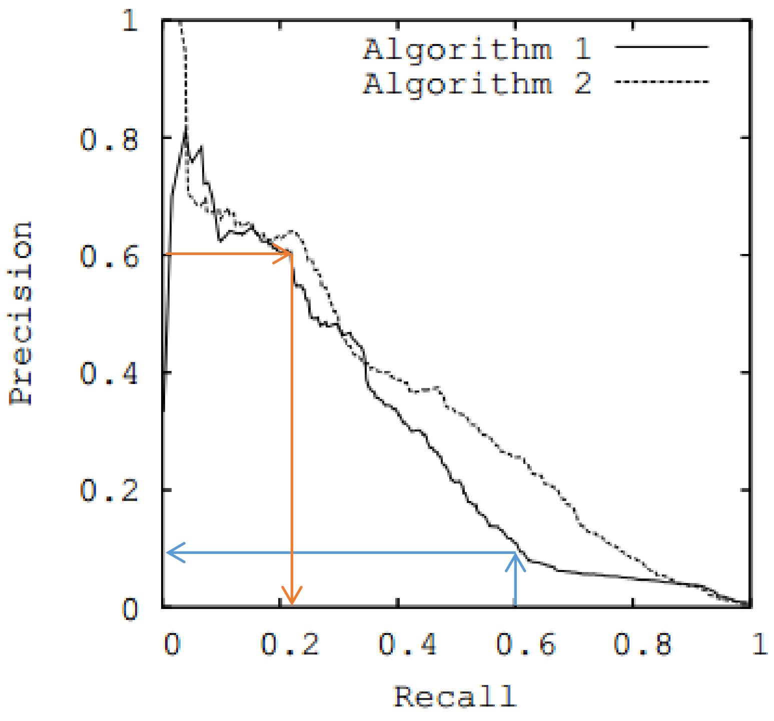


Figure 2.9 - Precision-Recall curves

The Precision-Recall curves associated with two different algorithms are presented in *Figure 2.9*. Focusing on the solid curve (i.e. "Algorithm 1"), if one modifies the threshold to obtain a recall equal to 0.6, the precision will be less than 0.2 (blue arrows in *Figure 2.9*). Similarly, if one wants a precision equal to 0.6, the recall will be less than 0.3 (orange arrows in *Figure 2.9*).

In the next section, the three classification models used in this thesis are introduced.

2.3.3. Models

The aim of a Machine Learning algorithm is to find a function (f) that well represents the relationship between inputs (features) and output (labels). The “Model” defines how the function is built and what are its main attributes. In Tensorflow.org (2020e) the model is defined as

a function with learnable *parameters that maps an input to an output. The optimal parameters are obtained by training the model on data. A well-trained model will provide an accurate mapping from the input to the desired output.*

Numerous models are available for addressing a classification problem. In this thesis, three different models have been used: Linear, Deep Neural Network and Wide&Deep.

2.3.3.1. The Linear model

In linear models, the relationship between the features and the labels is described as a linear function (Hastie, Friedman and Tibshirani, 2009):

$$Y = \beta_0 + \sum_{j=1}^{p} X_j \, \beta_j \tag{2.6}$$

being:

- $Y = labels$;
- $X = [X_1, X_2, \dots, X_p] = the\ features\ vector$;
- $X_j = a\ feature$;
- $\beta_0 = intercept\ (or\ bias)$;
- $\beta_j = coefficient\ (or\ weight)$;

The vector $\beta = [\beta_1, \dots, \beta_p]$ is the vector of *weights*. During the training, the optimal values of bias and weights are found. If a linear model is used in a binary classification problem with two features ($p = 2$ in equation 2.8), the decision boundary is a straight line. *Figure 2.10* clarifies this aspect.

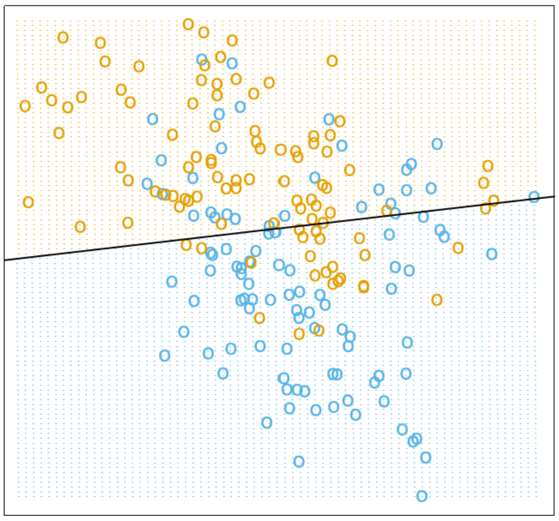


Figure 2.10 - Linear Regression of 0/1 Response (Hastie, Friedman and Tibshirani, 2009)

Figure 2.10 is a visual representation of a binary classification problem. "Orange" and "Blue" circles represent the real labels. The x and y-axis represent the values of the features (in this example only two features are considered). The solid black line represents the decision boundary generated by the linear model. The decision boundary divides the plane into two decision regions. Every circle above the decision boundary will be labelled by the model as "Orange". Every circle below the decision boundary will be labelled as "Blue". The number of wrong predictions (i.e. orange circles below the decision boundary and blue circles above the decision boundary) represents the False Negative and False Positive. Of course, the position of the decision boundary is strictly related to the threshold value (i.e. varying the threshold causes the boundary to translate).

It is worth noting that, in linear models, each feature is associated with a different coefficient (Hastie, Friedman and Tibshirani, 2009). In other words, each feature is independent, and the model cannot assess how "inter-features" relationships affect the output. This limitation can be partially solved by "*combining features into a single feature*" (TensorFlow.org, 2020c) and feeding this new, more meaningful, feature to the linear model; this process is called Feature Crosses (Google, 2020c). Still, the linear model is not able to generalize to previously unseen features combinations (Cheng *et al.*, 2016).

Despite its simplicity, the linear model is still widely used (James *et al.*, 2013); it is well-known, fast, reliable and it works well on large sets of features (Santini, 2018).

2.3.3.2. The Deep Neural Network model

In Deep Neural Network (DNN) models, the inputs (i.e. features) are linearly combined and converted into *derived features* through a non-linear function (Hastie, Friedman and Tibshirani, 2009). Derived features are named *hidden units*, and they constitute the so-called *hidden layer* of the Neural Network (Hastie, Friedman and Tibshirani, 2009). An example of a Neural Network with a single hidden layer is presented in *Figure 2.11*. Neural networks can have multiple hidden layers as well.

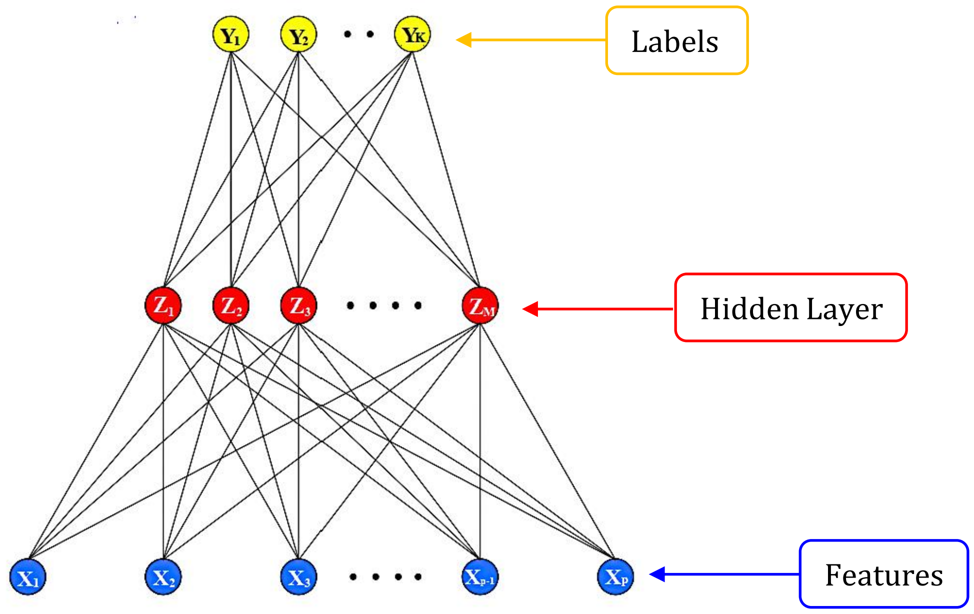


Figure 2.11 - Schematic of a single hidden layer, feed-forward neural network. Adapted from (Hastie, Friedman and Tibshirani, 2009)

In the figure above, the DNN model is fed with a vector of *p* features (X). Then, the features are linearly combined and converted into *M* derived features (Z) according to:

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X), \qquad m = 1, \dots, M \tag{2.7}$$

where:

- $\alpha_{0m} = bias;$
- $\alpha_m = vector\ of\ model\ coefficients;$
- $Z_m = derived\ feature\ (hidden\ unit);$
- $\sigma = activaction\ function.$

The activation function σ is non-linear. Usually, it is chosen to be the sigmoid function:

$$\sigma(v) = \frac{1}{1+e^{-v}} \qquad 2.8$$

If there is more than one Hidden Layer, equation 2.7 is used again to calculate the hidden units of the second hidden layer. In this case, X must be replaced by Z in eq. 2.7, and the coefficients must be updated.

Finally, the derived features of the last hidden layer are linearly combined and used to obtain the labels:

$$T_k = \beta_{0k} + \beta_k^T Z, \qquad k = 1, \dots, K \qquad 2.9$$

$$Y_k = g_k(T), \qquad k = 1, \dots, K \qquad 2.10$$

where:

- $T_k = linear\ combination\ of\ the\ derived\ features;$
- $\beta_{0k} = bias;$
- $\beta_k = vector\ of\ model\ coefficients;$
- $Z = [Z_1, \dots, Z_M] = vector\ of\ the\ derived\ features;$
- $Y_k = a\ label;$
- $T = [T_1, \dots, T_K];$
- $g_k = the\ output\ function.$

Several kinds of output functions exist. An example is the *softmax* function (Hastie, Friedman and Tibshirani, 2009):

$$g_k(T) = \frac{e^{T_k}}{\sum_{l=1}^{k} e^{T_l}} \qquad 2.11$$

The equations presented above refer to a general K classification problem (i.e. the number of labels to predict is K). For binary classification K = 2.

The number of hidden units (M), and the number of hidden layers, are adjustable parameters, and they can greatly affect the performance of the algorithm. In general, too many hidden units are better than too few (Hastie, Friedman and Tibshirani, 2009). The selection of the number of hidden layers is basically guided by experience and trial-and-error method (Hastie, Friedman and Tibshirani, 2009).

It is worth noting that the model has different sets of coefficients (also called *weights)*. When the labels (Y) are calculated, a set of K coefficients (β) and the bias (β_{0k}) are needed. Furthermore, each Hidden Layer requires the calculation of M coefficients (α_m) and a bias (α_{0m}). Similarly to the Linear model, the weights are optimized during the training phase.

The DNN model can overcome the limitations imposed by the Linear model. DNN models can capture nonlinearities in the data, they can produce decision boundaries of any shape and they can generalize better than the Linear model (Hastie, Friedman and Tibshirani, 2009; Cheng *et al.*, 2016). In *Figure 2.12*, the results of the application of a DNN model (on the same dataset described in *Figure 2.10*) are presented.

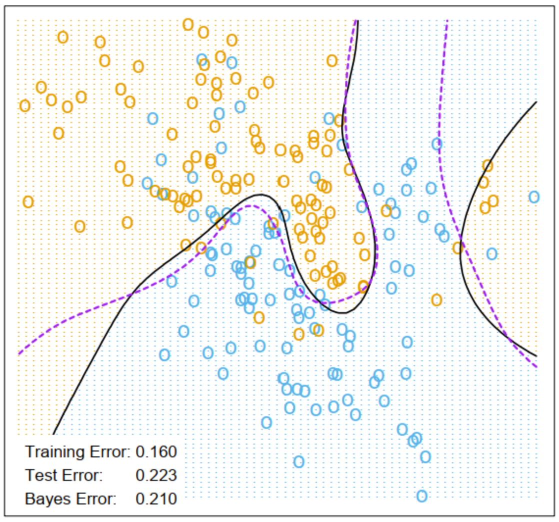


Figure 2.12 - Decision boundaries for a neural network model (Hastie, Friedman and Tibshirani, 2009)

The solid line in *Figure 2.12* represents the decision boundary produced by the DNN model. Comparing *Figure 2.12* with *Figure 2.10* reveals that the DNN model can produce more accurate and well-shaped decision regions.

DNN models are widely used in images and sounds recognition (Hastie, Friedman and Tibshirani, 2009; Brink, Richards and Fetherolf, 2016) and they are one of the most flexible models. Although, flexibility comes to a price: DNN requires more computational effort, they are harder to optimize and they are prone to overfitting (Hastie, Friedman and Tibshirani, 2009; Brink, Richards and Fetherolf, 2016).

2.3.3.3. The Wide&Deep model

The Wide&Deep model was created to join the benefits of the Linear (i.e. wide) and DNN models. As previously argued, the Linear model is fast, reliable and it is good at assessing the relative weight of each feature, or group of features (feature crosses). However, the linear model lacks in flexibility and generalization. Instead, the DNN model is flexible and better at generalizing and capturing inter-feature relationships and nonlinearities in the data. However, the deep model may overgeneralize and detect a relationship also where no (or poor) relationship exists.

To overcome the limitations of both models, and to enhance their qualities, the Wide&Deep model uses both the Linear and the Deep approaches (Cheng *et al.*, 2016). *Figure 2.13* depicts the general structure of the model.

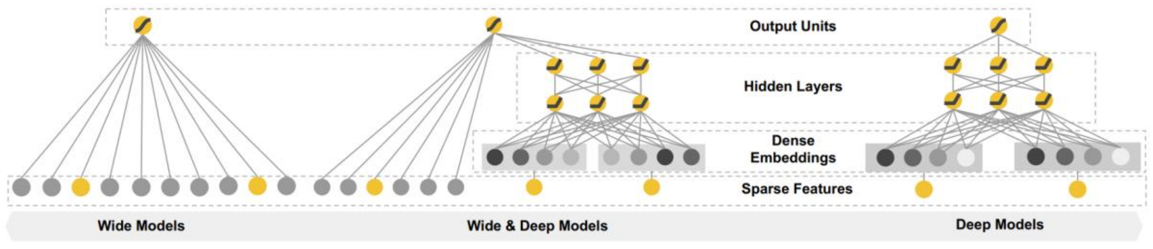


Figure 2.13 - The spectrum of Wide & Deep models (Cheng *et al.*, 2016)

The Wide&Deep model consists of a Linear part and a DNN part (centre of *Figure 2.13*). The two parts are jointly trained, and their parameters are optimized simultaneously (Cheng *et al.*, 2016). In the Cheng *et al.* (2016) original work, the wide part comprised only a small number of significant crossed features. The Wide&Deep model was used to develop a user recommender system, and it proved to perform better than the Linear and the Deep models.

2.3.4. TensorFlow

TensorFlow is an open-source, machine learning oriented, software library developed by Google Brain team and released under Apache 2.0 License in 2015 (Abadi *et al.*, 2016), the last stable version is r2.0.
The TensorFlow web homepage describes the software as:

> *an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications*.

TensorFlow is widely used for both research and production (Abadi *et al.*, 2016), and it owes part of its fame to its simplicity and flexibility. The library offers different levels of abstraction, including high-level APIs and pre-made estimators (i.e. a high-level representation of a complete model), which are particularly suitable for inexperienced users.
Several leading companies have used/use TensorFlow to solve real-world problems and increase productivity (TensorFlow.org, 2020b). Furthermore, the software is supported by an active and diverse online community (github, StackOverfolw).
TensorFlow runs on Ubuntu, Windows, MacOS and Raspberry (TensorFlow.org, 2020d). The python API is the most complete and easy to use, but also other languages are supported (e.g. C++, JavaScript, Go, and more) (GitHub.com, 2020; TensorFlow.org, 2020a).

In this thesis, TensorFlow r1.15 has been used, the platform has been installed on a python 3.7.4 release running on Windows 10.

Chapter 3
The alarm database

3.1. Introduction

The purpose of this chapter is twofold: firstly, to provide a brief description of the case study considered (the Yara production plant located in Ferrara - Italy; and secondly, to describe the alarm database used in this work.

3.2. The Yara production plant

The plant forms part of the chemical pole located in Ferrara (Italy), the main activity consists in the production of ammonia and urea. Due to the large quantity of hazardous substances stored and handled during normal activity, the plant has been classified as an “upper tier” Seveso III establishment, along with four more sites inside the chemical pole (Arpae, 2019).

The plant consists of seven different sections:

1. ammonia plant;
2. urea plant;
3. ammonia solution plant;
4. membrane and IGI plant;

5. carbon dioxide liquefaction plant;
6. AIR-1 plant (32.5% urea solution production);
7. urea storage and bagging plant.

Italian law ("Decreto Legislativo n. 105 del 26 giugno 2015 'Attuazione della direttiva 2012/18/UE relativa al controllo del pericolo di incidenti rilevanti connessi con sostanze pericolose.') (D.Lgs. 105/2015) requires that a Safety Report must be provided for "upper tier" establishments. Specifically, during the drafting of the Safety Report, a preliminary analysis shall be carried out, to identify critical sections of the plant (D.Lgs. 105/2015, Annex C, part 1). Two plant sections may meet the requirements to be considered critical, namely:

1. ammonia plant;
2. urea plant.

This result is strictly related to the substances handled in these two sections, as well as to the operating conditions.

Extensive use of methane, hydrogen and ammonia (anhydrous and aqueous solution) occurs in the ammonia plant section, while in the urea plant the key substances are ammonia, hydrogen, urea-formaldehyde (Formurea), and Sodium hypochlorite. A list of the main hazardous substances handled in the plant, along with their classification according to CLP regulation (1272/2008/CE) is presented in *Table C. 1*; bold tags represent hazardous properties subject to D.Lgs. 105/2015.
Table 3.1 summarizes what has been told so far about the hazardous substances present in the two plant sections of concern; it contains the hazardous characteristic that might lead to a major accident, the substances associated with those characteristics and the plant sections where these substances are produced/handled.

HAZARDOUS CHARACTERISTIC (subject to D.Lgs. 105/2015)	KEY SUBSTANCE	PLANT SECTION
Flammable gas	Anhydrous ammonia	• Ammonia plant • Urea plant
	Methane	• Ammonia plant
	Hydrogen	• Ammonia plant • Urea plant
Hazardous to the aquatic environment	Anhydrous ammonia	• Ammonia plant • Urea plant
	Ammonia aqueous solution (15 – 30 %)	• Ammonia plant
	Sodium hypochlorite (14 – 15 %)	• Urea plant
Acute toxicity	Anhydrous ammonia	• Ammonia plant • Urea plant
	Formurea	• Urea plant

Table 3.1 - Hazardous substances and plant sections

Remark: in *Table 3.1*, "key substance" has the meaning of "substance that best represents the hazardous properties of a stream/plant section". Therefore, the fact that methane can be found in the ammonia plant does not necessarily mean that it *is only found* in that plant section, or that methane *is the only* substance in that plant section. If a "key substance" is found within a specific plant section, one can conclude that there are streams or equipment whose hazardous properties are well described by the key substance.

In addition to an extensive use of both toxic and flammable substances, severe operational conditions (i.e. high temperature, and high pressure) are often required, due to the kinetic and thermodynamic features of the reactions involved. Furthermore, in specific parts of the plant, high temperature and pressure are associated with corrosive substances; this contributes to the risk of mechanical failure, which represents the major safety issue for this specific plant. Indeed, the technologies involved are extremely well known (they have been used for more than 60 years) and the SIS is efficient and reliable.
Therefore, the efforts must be directed to prevent:

- corrosion: caused by acid condensate in the ammonia plant and by carbamate and ammonium carbonate in the urea plant;
- Stress Corrosion Cracking: it arises in the anhydrous ammonia storage tanks, in presence of dissolved oxygen, high pressure and temperatures above 0°C;
- hydrogen embrittlement: worsened by high hydrogen concentration combined with high temperature (conditions that often arise in the ammonia plant section).

All the phenomena described above were considered during the design phase of the plant. Proper corrosion allowances, special alloys, lining, operational precautions and periodic non-destructive tests are the main safety barriers to avoid mechanical failure.

Even if both ammonia and urea plant sections are safety-critical, during the initial phase of this work the attention has been focused solely on the ammonia plant; this choice has been forced by the available data: Yara provided P&IDs and PFDs of the ammonia plant only.

The next paragraph presents an overview of the ammonia plant section, the purpose is to familiarize with the design and to describe the plant's operational activity. The equipment’s names have been changed due to the sensitive nature of this information.

3.2.1. The ammonia plant

The ammonia plant is based on Haldor Topsoe technology, and the capacity is 1720 t/d (Yara Italia S.p.A, 2016). Approximately 60% of the ammonia produced within this plant section is used to synthesize urea; the other part is sent to the Ravenna chemical pole via pipeline (Yara.it, 2020).

The plant section comprises seven subsections:

a. Desulfurization, Reforming and Breda Boiler;
b. Conversion, Decarbonization and Methanation;
c. Ammonia synthesis and Cooling circuit;
d. Anhydrous ammonia storage, Pipeline and Loading/unloading tankers.
e. Cooling and clarification towers;
f. Instruments air production and nitrogen compression and storage.

The ammonia synthesis is carried out according to the reaction:

$$N_2 + 3H_2 \leftrightarrow 2NH_3 \qquad 3.1$$

Nitrogen is supplied by air and hydrogen is produced through methane steam reforming.

Not all the sub-sections are equally interesting from the safety point of view. The analysis of past accidents (e.g. FACTS and MARS databases) that occurred in ammonia plants highlights that most of the accidents happened within the process section (i.e. subsections *a*, *b* and *c*). For this reason, and because of documentation availability constraints, only these three subsections are described in the next seven sub-paragraphs; each sub-paragraph describes the design and the activities of the sub-section of concern. The only exception is the Breda boiler: it has been decided to not further describe this unit because it is not directly involved in the ammonia production.

3.2.1.1. Desulfurization

Methane enters the plant via pipeline in the form of natural gas $\left(vol\%_{CH_4} \approx 98\ \%\right)$, the pressure is then reduced from 50 bar to 40 bar, which is the feed pressure. Even though the natural gas content of sulphur compounds is relatively low, the catalyst used in the upcoming sections of the plant is extremely sensitive to these compounds. Consequently, sulphur must be removed to avoid the risk of catalyst deactivation.
Desulfurization section (first step of sub-section *a.*) consists of three catalytic reactors:

1) R – 03: hydrogenator;
2) R – 01: first sulphur absorber;
3) R – 02: second sulphur absorber.

After a preheating stage, the natural gas at the temperature of 400°C flows into R – 03; in the reactor, the sulphur compounds are hydrogenated to H_2S according to the reactions:

$$RSH + H_2 \rightarrow RH + H_2S \qquad 3.2$$

$$R_1SR_2 + 2H_2 \rightarrow R_1H + R_2H + H_2S \qquad 3.3$$

$$COS + H_2 \rightarrow CO + H_2S \qquad 3.4$$

Then, the gas flows into R – 01 and R – 02 reactors (usually arranged in series) where hydrogen sulphide is absorbed according to the reaction:

$$H_2S + ZnO \rightarrow ZnS + H_2O \qquad 3.5$$

Finally, the natural gas flows out the sub-section with a sulphur content < 1 ppm, and it approaches the Reforming sub-section.

3.2.1.2. Reforming

The de-sulphurated natural gas is mixed with medium pressure steam ($P \approx 40\ bar$) to ensure an optimal vapour/carbon feed ratio ($Steam/C\ \approx 3$). Next, the mixture is preheated and enters the reforming section (second step of sub-section *a.*) at a temperature of 540°C.
The reforming section comprises two reforming stages:

1) B – 01: primary reformer;
2) R – 03: secondary reformer.

The primary reformer is a vertical, proprietary, side-fired reactor where the steam reforming of methane takes place according to the reactions:

$$CH_4 + 2H_2O \leftrightarrow CO_2 + 4H_2 \qquad 3.6$$

$$CO_2 + 2H_2 \leftrightarrow CO + H_2O \qquad 3.7$$

Due to the endothermic nature of the reactions involved, as well as their thermodynamic and kinetic properties, the temperature inside the reactor must rise to 800°C; to achieve this goal, a heat source is needed (Aika *et al.*, 2012). In Haldor Topsoe based reformers, natural gas is burned to provide heat and maintain the desired temperature level. Specifically, in this reformer, natural gas is burned in more than six hundred axial burners. Subsequently, heat is recovered from the exhaust gas leaving the combustion chamber and it is used to pre-heat the reacting mixture, to pre-heat the natural gas entering the desulphurization section, to pre-heat the air stream entering the secondary reformer, to superheat steam and to pre-heat a boiling feed water stream.
The reacting mixture (methane and steam) flows from the top to the bottom of the primary reformer, in more than 400 catalytic tubes. These tubes are placed inside the reformer's combustion chamber; the heat from the outside promotes the steam reforming according to reactions 3.6 and 3.7.
The mixture leaves the primary reformer ($T_{out} \approx 800°C$) and approaches the secondary reformer.

The secondary reformer (*Figure 3.1*) is an autothermal, adiabatic reactor. Two streams enter the reformer: the process streams and an air stream. The aim of this stage is to lower the methane content, to increase the hydrogen content and to add nitrogen.

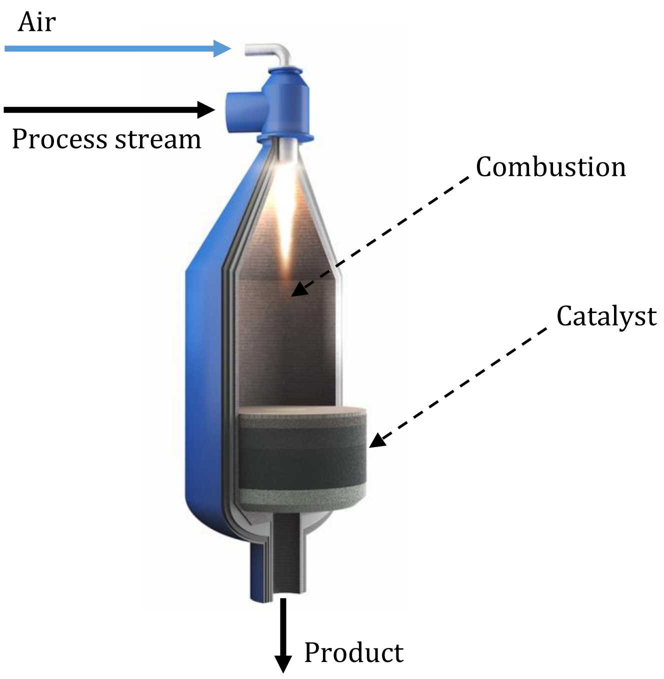


Figure 3.1 - Secondary reformer representation. Adapted from (Topsoe.com, 2020)

In the first section of the reactor, the two streams are mixed, and a combustion reaction occurs (air oxidises part of the unreacted hydrocarbons and part of the hydrogen in the process stream) according to the reactions (AL-Dhfeery and Jassem, 2012):

$$CH_4 + 3/2\, O_2 \leftrightarrow CO + H_2O \qquad 3.8$$

$$H_2 + 1/2\, O_2 \leftrightarrow H_2O \qquad 3.9$$

The partially reacted mixture flows through the second section of the secondary reformer, where a catalytic bed promotes the reactions (AL-Dhfeery and Jassem, 2012):

$$CH_4 + H_2O \leftrightarrow CO + 3H_2 \qquad 3.10$$

$$CO + H_2O \leftrightarrow CO_2 + H_2 \qquad 3.11$$

$$CH_4 + 2H_2O \leftrightarrow CO_2 + 4H_2 \qquad 3.6$$

Reaction 3.10, 3.11 and 3.6 are endothermic, the required heat is provided by the initial combustion (reactions 3.8 and 3.9). The temperature inside the reactor exceeds 1000°C.
The resulting mixture leaves from the bottom of the secondary reformer with a composition (dry basis) equal to:

- $H_2 = 55\ \%$
- $CO = 24\ \%$
- $CO_2 = 8\ \%$
- $CH_4 = 0.5\ \%$

The process stream leaving the reforming section needs to be purified from carbon monoxide and carbon dioxide before entering the ammonia synthesis section. The removal of CO and CO_2 is carried out in subsection b through three subsequent steps. First, carbon monoxide is converted into carbon dioxide and hydrogen in the conversion section. Then, CO_2 is removed in the decarbonization section. Finally, the last traces of CO and CO_2 are removed in the methanation section.

3.2.1.3. Conversion

After leaving the secondary reformer, the gas stream enters the conversion section (first step of sub-section *b.*) where the CO conversion is carried out. The main equipment is:

1) E – 01A/B: heat recovery reboilers;
2) E – 02: steam superheater;
3) R – 04: high-temperature conversion reactor;
4) R – 05: low-temperature conversion reactor;
5) E – 03: heat recovery reboiler.

Heat is removed in two boiling feed water reboilers (arranged in parallel) and in a steam superheater (E – 02); the temperature of the syngas stream leaving the cooling section is 340°C.
The cooled gas stream passes through the two catalytic reactors (R – 04 and R – 05) where the partial conversion of CO occurs according to the exothermic reaction:

$$CO + H_2O \rightarrow CO_2 + H_2 \qquad 3.12$$

A cooling stage is located between the two reactors. Here, heat is removed in E – 03 and in three boiling feed water preheaters arranged in series (not mentioned in the list of the main equipment).

Finally, the process stream leaves the conversion section at a temperature of 220°C, with a CO content of 0.2 % (dry basis) and a CO_2 content equal to 17.6 % (dry basis).

3.2.1.4. Decarbonization

The gas stream leaving the conversion section has a low content of CO while the content of CO_2 is high. Carbon dioxide needs to be removed before proceeding further; this is achieved in the decarbonization section (second step of subsection *b, Figure 3.2*). The main equipment is:

1) C – 01: absorption column (chemical absorption);
2) C – 02: regeneration column;
3) miscellaneous pumps and heat exchangers;

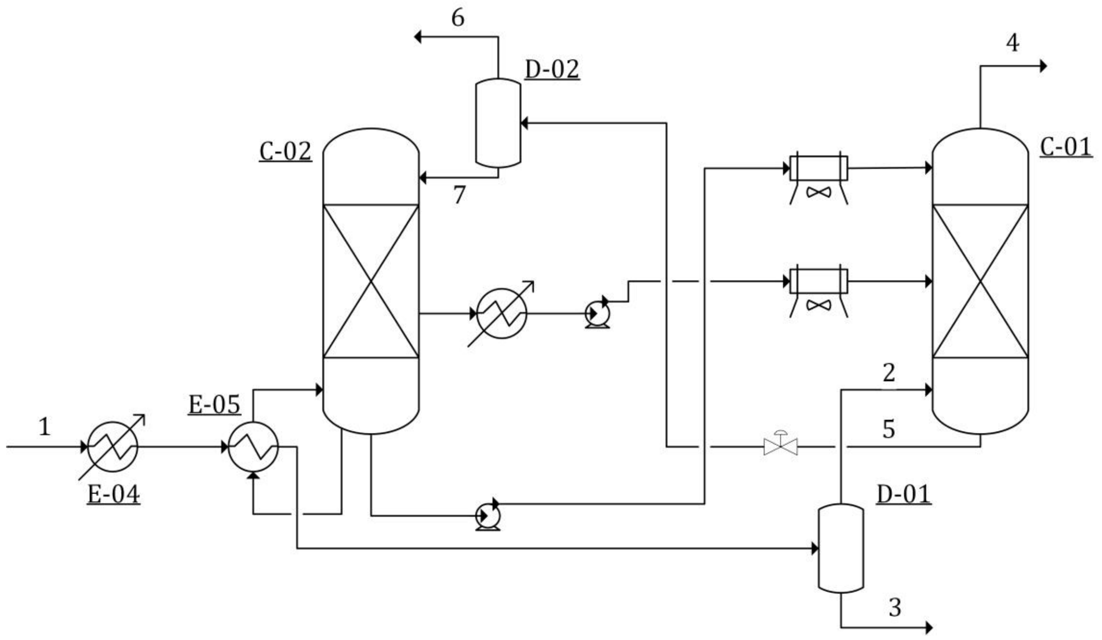


Figure 3.2 - Decarbonization section

Gas absorption, as is well known, is enhanced by low temperatures. Consequently, before entering the absorption column, the process stream (stream 1 in *Figure 3.2*) is

cooled in three heat exchangers. In the first cooler (E – 04), heat is used to produce steam. The second and third exchangers (E – 05, arranged in parallel) are the reboilers of the regeneration column.
Then, process condensate (3) is removed in D – 01, and the cooled gas stream (2) enters the absorption column (C – 01), where a Vetrocoke solution (containing K_2CO_3, $KHCO_3$, DEA and glycine) is used as a solvent. Here, CO_2 is absorbed according to the reaction:

$$CO_2 + K_2CO_3 + H_2O \rightarrow 2KHCO_3 \qquad 3.13$$

The clean process gas (4) leaves the top of column C – 02 with a carbon dioxide content of 700 ppm and a temperature of 90°C.
The (CO_2 rich) liquid solution (5) flows from the bottom of C – 01 to the top of the regeneration column (C – 02). Here the liquid flashes (pressure decreases from 30 bar to 2 bar in D – 02) and the released gas (CO_2 reach) leaves section (6). The fraction of the liquid solution that remains liquid after the flash (7) flows down the regeneration column, eventually reaching the bottom and the reboilers. The heat provided by the reboilers and the previous flash promote the inverse of 3.13 reaction, the result is CO_2 liberation and, consequently, solvent regeneration.

3.2.1.5. Methanation

The clean syngas leaving the top of the absorption column approaches the methanation section (third step of subsection *b.*) that essentially consists of:

1) R – 06: the methanation catalytic reactor;
2) miscellaneous heat exchangers.

The process stream is preheated from 90°C to 220°C end enters the reactor, where all the remaining traces of CO_2 and CO are removed according to the reactions:

$$CO + 3H_2O \leftrightarrow CH_4 + H_2O \qquad 3.14$$
$$CO_2 + 4H_2O \leftrightarrow CH_4 + 2H_2O \qquad 3.15$$

The gas stream leaving the reactor has an inorganic carbon content (CO_2 and CO) of less than 10 ppm, and a temperature of 245°C (3.14 and 3.15 are exothermic reactions). Finally, heat is recovered, and the temperature is lowered to 30°C. Before the cooling section, there is an emergency line to flare stack.

3.2.1.6. The ammonia synthesis

The clean and cooled syngas has now the required purity for ammonia production. Nevertheless, ammonia synthesis reaction (3.1) is exothermic and reversible and requires high temperature to ensure fast kinetic and high pressure to ensure high conversion (Aika *et al.*, 2012; Pattabathula and Richardson, 2016). Thus, the reacting mixture needs to be compressed and heated before it enters the reactors. Furthermore, due to the reversible nature of 3.1, the conversion is only partial, and a considerable amount of unreacted syngas leaves the reactors. The ammonia needs to be separated from the unreacted syngas, that is eventually recycled back to the reactors (*Figure 3.3*). The separation is obtained through subsequent cooling and expansion units where ammonia liquefies, and the generated gas is either recycled back to the reactors or purged to permit the inert compounds to leave the reaction loop. Furthermore, in the cooling section, the process stream needs to be cooled below 0 °C, a refrigerant is then needed. To achieve this task part of the produced ammonia is used as a refrigerant in a refrigeration loop, where anhydrous ammonia is compressed and evaporated in three different stages.

The ammonia synthesis section (first step of subsection *c.*) consists of:

1) P – 01: syngas compressor;
2) R – 07 and R – 08: ammonia synthesis catalytic reactors (proprietary);
3) E – 06: heat recovery reboiler;
4) P – 02: cooling circuit compressor;
5) miscellaneous exchangers and equipment (including a liquid ammonia washing section)

P – 01 is a five-staged, steam-driven, centrifugal compressor with inter-stage cooling units. Before the first stage, and between the first and second stages, the cooling units consist of liquid ammonia chillers (ammonia from the refrigeration circuit is used as a refrigerant); in the remaining cooling units, water is used as a refrigerant. After the fourth stage of the compressor, the reaction mixture is sent to a washing section to remove any trace of the oxygenated compound (CO, H_2O) and to further decrease the temperature. Here, the gas is cooled in two heat exchangers and is sent to column C – 03, where it comes into contact with liquid ammonia; this ensures the oxygenated compounds elimination (absorbed by ammonia) and an outlet temperature of 2°C. The stream leaving the washing unit is sent to the fifth (and final) stage of the compressor. The process stream enters the compressor at 25 bar and leaves at 180 bar.

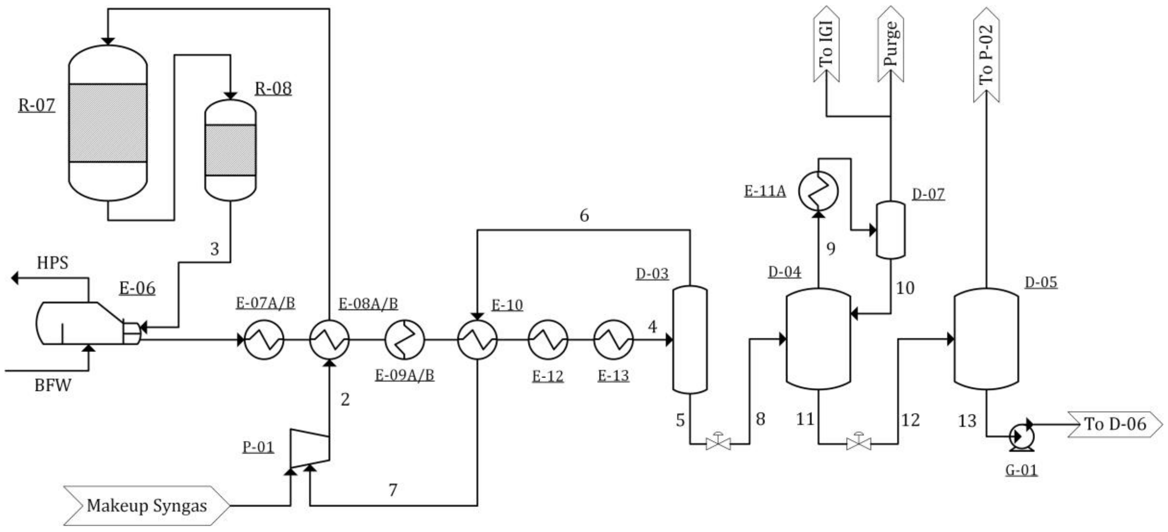


Figure 3.3 - Ammonia reaction loop

The reacting mixture that leaves the fifth stage of P – 01 (makeup) is mixed with the unreacted syngas (stream 7 in *Figure 3.3*), which comes from the synthesis loop. The resulting stream (2) is preheated in E – 08 A/B (gas-gas heat exchangers, they recover heat from the hot stream leaving the reactors) and sent to the first synthesis reactor. In R – 07 and R – 08 the exothermic reaction 3.1 occurs, the generated heat is then recovered and used to produce high-pressure steam in E – 06 reboiler, to preheat the reacting mixture entering R – 07 and to preheat boiling feed water.

The product stream (3) leaves the reactors with a temperature of 400 °C and a composition equal to:

- $H_2 = 50\ \%$
- $N_2 = 18\ \%$
- $NH_3 = 22\ \%$
- $CH_4 = 7\ \%$
- $Ar = 3\ \%$

As already mentioned, the product stream contains a significant amount of reagents (H_2 and N_2) that must be recycled to the reactors, as well as inert compounds (CH_4 and Ar) that must be purged. This is achieved by cooling and expanding the product stream.

The cooling section consists of ten heat exchangers arranged in series. The first, second and third heat exchangers (E – 06 and E – 07A/B) are water-cooled, the fourth and the fifth are a gas-gas heat exchangers (E – 08A/B), the sixth and the seventh are water-cooled (E – 09A/B), the eight is a gas-gas heat exchanger (E – 10) the ninth and the tenth coolers are liquid ammonia chillers (liquid ammonia comes from the refrigeration circuit). The result is that the temperature of the process stream is lowered below the mixture's dew point; this leads to the formation of a liquid phase (ammonia-rich). The biphasic stream (4) enters a separator (D – 03) where the liquid phase is collected and sent to the expansion section (5). The gas-phase (reagents rich) leaves from the top of the separator (6), enters E – 10 and, finally, is recycled back to the compressor (7) where is mixed with the make-up stream leaving the fifth stage of P – 01; this completes the reagents loop.

At the same time, the liquid phase leaves from the bottom of D – 03 (5) and enters the expansion section at 172 bar and -2°C. The expansion occurs in two vessels arranged in series where the liquid stream is expanded (and partially vaporized) from 170 bar to 1.3 bar. In the first vessel (D – 04), the product stream expands from 170 bar to 20 bar (8), part of the liquid vaporizes, and the resulting gas (9) is sent to an ammonia chiller (E – 11A) where the ammonia is condensed and recycled back (10) to D – 04. On the other hand, the non-condensable fraction of the gas is sent part to purge and part to the membrane and IGI plant section. Next, the liquid phase (11) is sent, form the bottom of D – 04, to the second expansion vessel (D – 05) where it expands from 20 bar to 1.3 bar (12), part of the liquid vaporizes and the resulting gas is sent to the compressor of the refrigeration cycle. The liquid phase collected in the bottom of D – 05 is anhydrous ammonia at -30 °C (temperature has lowered due to the expansion) which is sent (13) to the cryogenic storage vessel (D – 06), this ends the ammonia reaction loop.

As already mentioned, a refrigerant is needed to cool the product stream leaving the synthesis reactors (liquid ammonia chillers E – 12 and E – 13), to cool the makeup stream entering the fifth stage of P – 01 (ammonia chiller E – 14) and to condense and recover the ammonia from the purge gas streams (E – 15, E – 11A/B). For this reason, part of the produced ammonia is used as a refrigerant in a refrigeration loop.

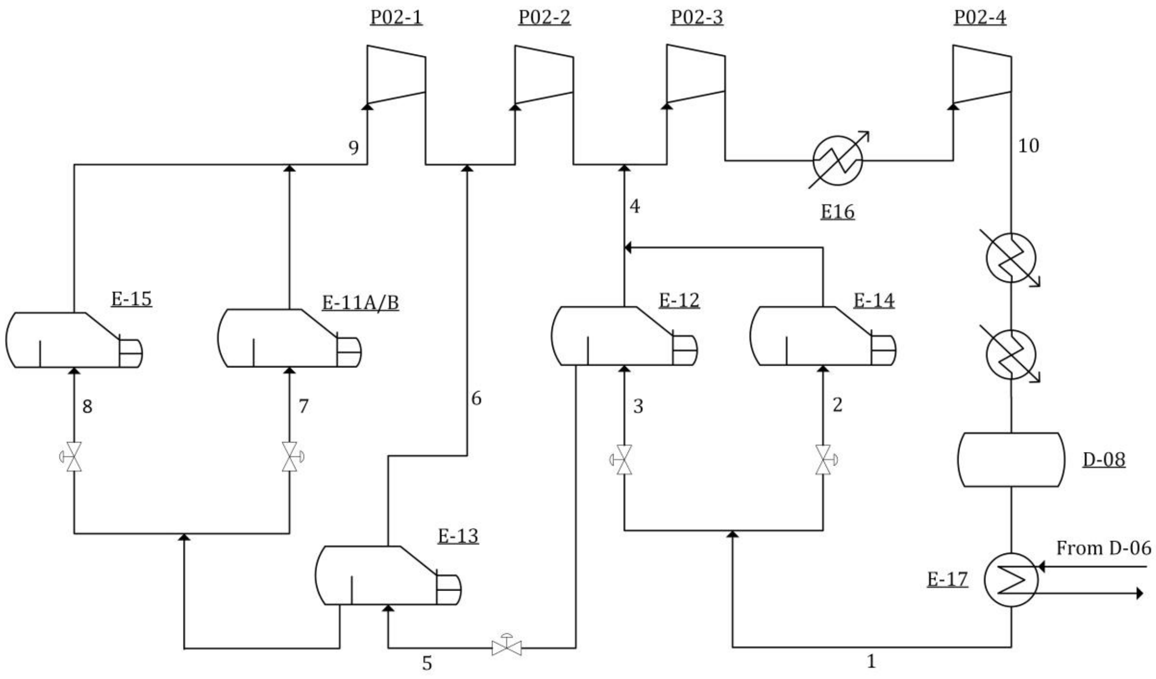


Figure 3.4 - Ammonia refrigeration cycle

Here, according to *Figure 3.4*, the cool stream of liquid ammonia (stream 1) is expanded and subsequently evaporated in three stages, at three different pressures, corresponding to the temperatures of 4°C (ammonia at 4 bar, stream 2 and 3), -7°C (ammonia at 2 bar, stream 5) and -30°C (ammonia at 0.05 bar, streams 7 and 8). After the evaporation, ammonia is sent to P – 02, which is a centrifugal, 4 staged, compressor. Each vapour stream enters a different stage of the compressor depending on the pressure: the ammonia streams flowing from E – 12 and E – 14 (stream 4) enter the third stage of the compressor, the one flowing from E – 13 (stream 6) enters the second stage of the compressor and the ones coming from E – 15 and E – 11A/B (stream 9) enters the first stage of the compressor. Between the third and the fourth stage of the compressor, the gas stream is cooled in E – 16. The ammonia leaves the compressor at a pressure of 20 bar (stream 10). Next, the ammonia is condensed and sent to an accumulator tank (D – 08). From the accumulator tank, the refrigerant flows through E – 17, where is cooled by the anhydrous ammonia coming from the cryogenic storage vessel (D – 06). Finally, the cooled refrigerant (1) enters again the ammonia chillers, and the refrigeration loop is completed.

The description of the ammonia plant section is now concluded. The hope is that the main features and safety-related issues have been clarified, although the description has not covered all the aspects and details of the plant. More about the ammonia synthesis and, specifically, about Haldor Topsoe technology can be found in Jennings (1991), Aika *et al.* (2012), Pattabathula and Richardson (2016). More about the Ferrara production site can be found in (Yara Italia S.p.A, 2016) and (Yara.it, 2020).

The next paragraph focuses on the alarm database provided by Yara. It contains alarm data coming from the plant described above, and it represents the basis on which all the analyses performed during this thesis work have been built.

3.3. The alarm database

The database consists of alarm data collected during an observation period of more than four months; specifically, from 19/07/2017 to 30/11/2017. Each row of the database (26473 in total) represents an alarm occurrence, and each column (thirty-six in total) represents a piece of information about the alarm. Thus, for each alarm occurrence, thirty-six alarm attributes are provided. Not all these attributes are equally important or meaningful, some of them are just codes to identify the different plant sections, the stations, or the GMT time of the station. According to (Kondaveeti *et al.* (2010) an alarm is completely defined by three attributes only:

- Time Stamp: the time when the alarm occurred;
- Source: the instrument or the PLC function that triggered the alarm;
- Alarm Identifier: defines the alarm status (e.g. HHH, LLL, HTRP, etc.)

In *Table 3.2*, a list of the most meaningful alarm's attributes is presented; for each attribute, a brief explanation of its meaning is provided.

ATTRIBUTE	MEANING
Time Stamp	Date and time (GMT) of the alarm occurrence.
Source	The source that triggered the alarm. It might be a measuring instrument or a PLC function.
Jxxx	The safety interlock logic associated with the alarm. When an alarm is triggered, the corresponding safety logic is activated. The logic initiates a series of predefined actions, that might be as simple as turning a pump on, or as complex as tripping of multiple pieces of equipment.
Message	The message that is shown to the operator. It contains five attributes: 1. the Source; 2. a concise description of the equipment involved; 3. the safety interlock logic (Jxxx); 4. the value and units of measures of the process variable; 5. the Alarm Identifier.
Active Time	Date and time (GMT) of the first alarm occurrence. If the alarm is not a Recover entry, this field is equal to "Time Stamp".
Data Value	The value of the process variable. If the "Source" is not a measuring instrument this field is empty.
Eng. Unit	The units of measure of the process variable. If the "Source" is not a measuring instrument this field is empty.

Table 3.2 - Alarm database attributes

A few more words are needed to describe the Alarm Identifier attribute (point *5* of attribute Message). As it has been already mentioned, the Alarm Identifier defines the alarm status. In the alarm database, twelve different Alarm Identifiers can be found:

- LLL / HHH
  low/high alarm. It warns that the value of the process variable (or PLC function) has exceeded the low/high setpoint. It announces that a block intervention will occur if the value continues to decrease/increase.

- LTRP / HTRP
  very low/very high alarm. It informs the operator that the value of the process variable (or PLC function) has exceeded the very-low/very-high setpoint. Usually, after this alarm, a block intervention starts (other conditions might be needed to start the block procedure).

- LLL / HHH / LTRP / HTRP Recover
  recover of a previous LLL/HHH/LTRP/ HTRP alarm. It informs the operator that the alarm has been recovered.

- IOP
  instrumental failure or out-of-range measure.

- ALM
  generic alarm (used for alarms triggered by ad-hoc logics).
- NR
  alarm terminated. It informs the operator that an IOP or an ALM is terminated.

- ACK
  acknowledgement. It informs that the operator has acknowledged the alarm (typically pressing a button). Acknowledgement might be required by/given to any of the Alarm Identifiers presented above.

This completes the overview of the database's structure and keywords. In the next paragraph, the results of the analysis carried out by (2018) are presented, which are necessary to achieve a better understanding of the database main features and data distribution.

3.3.1. Database analysis

The database contains 26473 alarms. Obviously, the alarms are not evenly distributed throughout the observation period. The time-distribution of the alarms is presented in *Figure 3.5*.

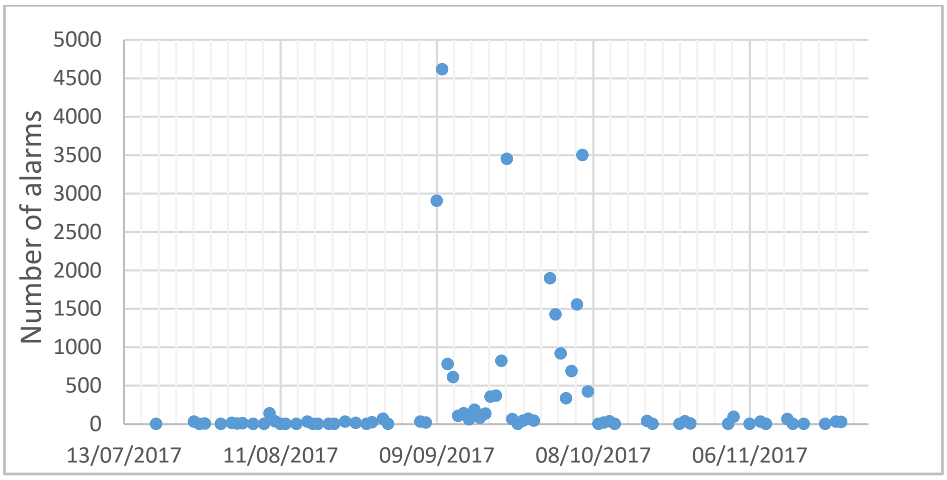


Figure 3.5 - Alarms time distribution

The days between the 9th of September and the 8th of October show an unusually high alarm count compared with the rest of the observation period. One can conclude that the plant went through a period of high instability during these days. Specifically, 25572 alarms occurred during that month (more than 96 % of the total alarm registered in the database). Clearly, a considerable number of floods and chattering alarms must have occurred.

The plant instability is also suggested by the process variables time-trend, an example is presented in *Figure 3.6*.

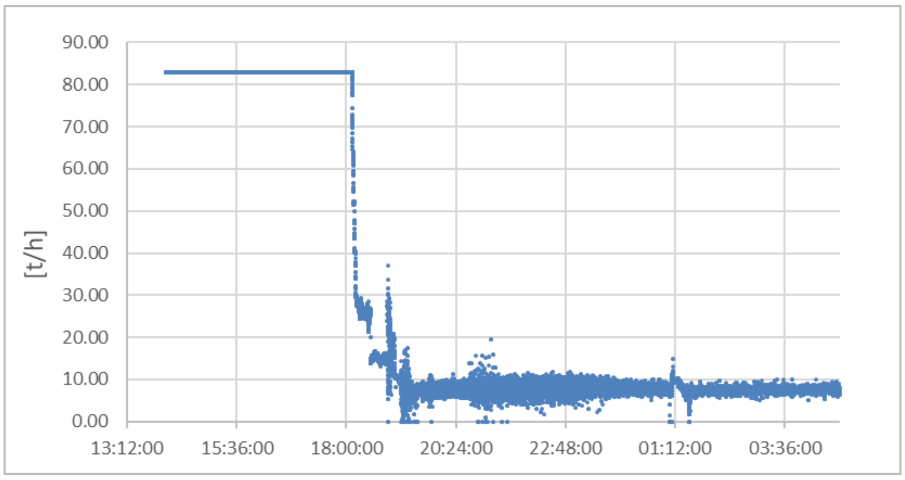


Figure 3.6 - FI209B time-trend on the 9th of September 2017

The figure shows the data measured by the instrument FI209B during the 9th of September (it measures the mass flow rate of steam entering the first reformer); the x-axis represents the time of the day. Undoubtedly, an event caused a significant reduction of the flow at 6 pm; every process variable in the ammonia plant section shows the same trend. The plant instability was caused by a total power outage, happened on the 9th of September at 6 pm, as reported in the document "Analysis of operational experience" provided by Yara. The blackout forced a plant shut down and a small quantity of ammonia (150 kg) was released into the atmosphere due to the energy loss in the cooling system.

The alarms registered during the observation period were triggered by 194 different sources in total. Clearly, the alarm count is not evenly distributed between the sources (i.e. not all the sources have the same alarm count). In *Figure 3.7* the twenty alarm sources with the higher alarm count are presented.

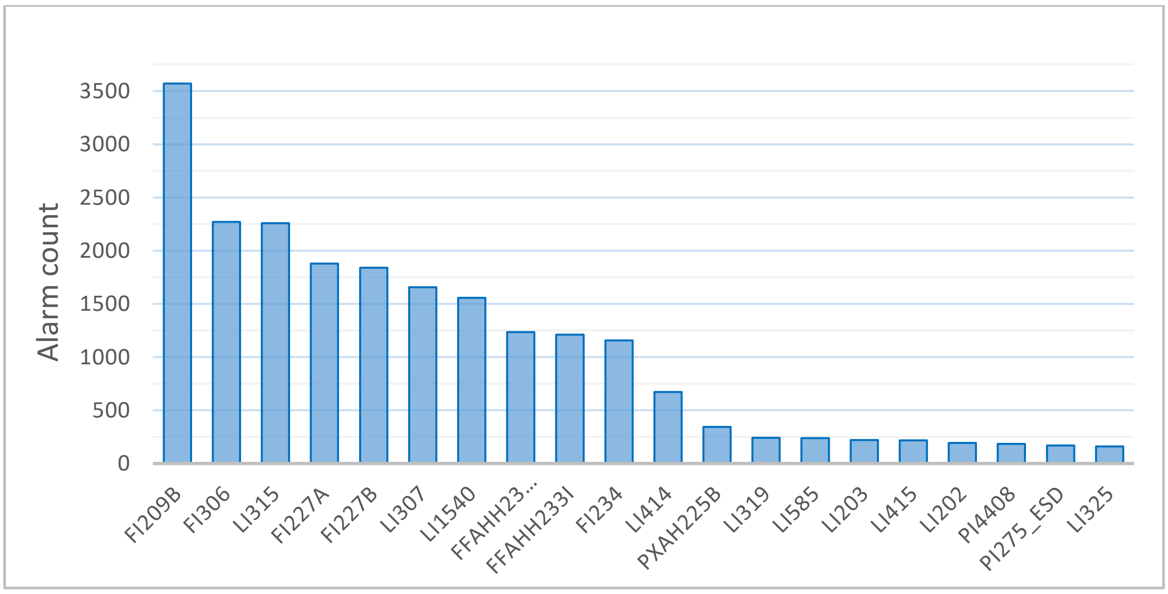


Figure 3.7 - Top 20 sources with higher alarm count

It is worth noting that the first ten sources show an extremely higher alarm count compared to the others. Specifically, 21269 alarms were triggered by the first ten sources in *Figure 3.7*. Thus, the considerations made in paragraph *2.2.3* point *4* match the data evidence. Efforts must be directed to assess the features and the behaviour of the alarms with higher alarm count (e.g. do they show chattering? Are they redundant? Etc.) and, whenever possible, to decrease the occurrences of these "bad actors" since this would significantly improve the performance of the alarm system and the operator response.

3.3.2. Alarm locations and functions

To achieve a better understanding of the alarm system, it would be useful to know where the measure instruments (sources) that triggered an alarm are in the ammonia plant, and how the safety interlock system acts. This would clarify the relation between alarms, equipment and PLC safety functions; which is not a trivial task looking at the alarm database only. Indeed, the only information provided in the alarm database about the location and the safety function is in the "Message" attribute (*Table 3.2*). Although, the provided information is concise and often difficult to interpret. For example, *Table 3.3* represents a row of the alarm database (the most meaningful attributes only are presented):

Time Stamp	Source	Jxxx	Message	Active Time	Data Value	Eng Unit
----	LI202	J1/J2	LI202 D201 LEVEL J1/J2 LTRP	----	0.0	%

Table 3.3 - A row of the alarm database (reduced)

Looking at the attributes in *Table 3.3* one can conclude that LI202 is a level gauge that measures the level of D201 vessel. But it is still not clear neither where D201 is located nor the function of the piece of equipment. Similarly, it is not clear the function of the J-1 and J-2 interlock functions after being triggered. For this reason, it has been tried to find the location of the alarm sources. Fifty-six sources out of one hundred ninety-four have been found. The reasons for which the remaining sources have not been found may be the following:

- The alarm is not associated with an instrument;
- The alarm is not located in the Ammonia plant section.

The results of the "Search operation" are presented in *Table C. 2*.

Chapter 4

Analysis set-up

4.1. Introduction

In the present chapter, the analyses performed during the thesis work are presented and described in detail. Specifically, it is clarified how the binary alarm database has been obtained from the original alarm database. Then, the steps to be followed to obtain the HDAP, the ASCM and the Chattering Index are described. Later, the idea of Dynamic chattering assessment is introduced, and how to obtain the related index is clarified. Finally, the step-by-step procedure that has been followed to build the Machine Learning models is outlined.

It is worth noting that the analyses have been performed using a reduced version of the alarm database; specifically, from 09/09/2017 to 08/10/2017. This has been done both for time constraints (the analyses are extremely time-consuming) and because most of the alarms occurred within that period.

4.2. The binary database

As previously argued (*3.3*), an alarm event is uniquely identified by three attributes:

1. *time stamp;*
2. *tag name;*
3. *alarm identifier.*

The combination of a *tag name* and an *alarm identifier* (e.g. FI209B HHH, FI234 LTRP etc.) is called a *unique alarm* (Kondaveeti *et al.*, 2010). Each unique alarm data can be conveniently represented by a binary sequence.
The binary sequence associated with a unique alarm is a vector whose elements can be "0" or "1". Each element of the vector is associated with a *time stamp*; normally, a one-second sampling is required (i.e. the elements of the vector are one-second-spaced). A "0" in the binary sequence means that, at that instant, the unique alarm of concern has not occurred. On the other hand, a "1" in the binary sequence means that the unique alarm has occurred at that instant.
The binary sequences obtained by each of the unique alarms can be grouped and displayed as a matrix (the *binary database*). The binary representation of the alarm data represents the starting point for each advanced alarming technique described in the next three sections (*4.3*, *4.4* and *4.5*). Now, the steps to obtain the binary database from the Yara alarm database (section *3.3*) are described.

The binary database has been obtained through three steps:

1. from the alarm database, all the unique alarms that occurred within the observation period have been identified. Then, the unique alarms have been stored in a vector (each element of the vector is a unique alarm). 791 unique alarms have been identified during this step;
2. a one-second-spaced time vector has been built. The first element of the vector represents the starting point of the observation period while the last element of the vector represents the ending point. The original database (section *3.3*) contains alarm data from 19/07/2017 to 30/11/2017. In this thesis, it has been decided to focus on the period between 09/09/2017 and 08/10/2017

only, because most of the alarms occurred within that time span (*Figure 3.5*). *Table 4.1* is a representation of the obtained time vector.

09/09/2017 00:00:00
09/09/2017 00:00:01
09/09/2017 00:00:02
09/09/2017 00:00:03
…
08/10/2017 23:59:58
08/10/2017 23:59:59
09/10/2017 00:00:00

Table 4.1 - One-second-spaced time vector

The time vector has 2592000 elements (i.e. seconds between 09/09/2017 and 09/10/2017);

3. for each unique alarm identified in step *1*, a binary vector has been built (same length as the time vector). Each element of the binary vector is a "1" if the unique alarm occurred at the moment identified by the element of the time vector or is a "0" if it did not occur.

As a result, a 2592001 x 792 matrix has been obtained, representing the binary sequences of all the unique alarms that occurred within the period of concern. The first column of the matrix is the time vector, the remaining columns are the binary sequences. The first row of the matrix represents the unique alarms.
The obtained matrix is large and difficult to handle. However, some columns and rows contain only null elements. If a row contains zeroes only, no alarm occurred at the instant defined by the first element of the row (i.e. an element of the time vector). If a column contains zeroes only, the associated unique alarm has not occurred within the period of concern. For this reason, the rows and the columns containing zeroes only have been removed. As a result, a reduced 18876 x 764 matrix has been obtained. *Table 4.2* provides a schematic representation of the binary database.

Time stamp	ADAH3400A_8 ACK	…	FI306 LLL	…	PI3400 ACK	…	ZAL320 ALM
09/09/2017 16:07:24	0	…	1	…	0	…	0
09/09/2017 16:07:25	0	…	0	…	0	…	0
09/09/2017 16:07:26	0	…	0	…	0	…	0
…	…	…	…	…	…	…	…
09/09/2017 16:07:39	0	…	0	…	0	…	1
…	…	…	…	…	…	…	…
09/09/2017 16:15:54	0	…	1	…	0	…	0
…	…	…	…	…	…	…	…
21/09/2017 09:10:42	1	…	0	…	0	…	0
…	…	…	…	…	…	…	…
07/10/2017 13:57:13	0	…	0	…	0	…	0
07/10/2017 14:25:41	0	…	0	…	1	…	0

Table 4.2 - Schematic representation of the binary database

It is worth noting that the first row of the matrix is associated with the time stamp "09/09/2017 16:07:24". Thus, from the beginning of the period under assessment (i.e. "09/09/2017 00:00:00", the first element of *Table 4.1*) until "09/09/2017 16:07:24" no alarm occurred. Similarly, no alarm occurred between the time stamp associated with the last row of the matrix (i.e. "07/10/2017 14:25:41") and the end of the period under assessment ("09/10/2017 00:00:00", last element of *Table 4.1*). Thus, the observation period can be further reduced, and it can be defined as the time span between the first row of *Table 4.2* and the last row of the same table (i.e. from 09/09/2017 16:07:24 to 07/10/2017 14:25:41).

The reduced binary database has been used as a starting point to obtain the HDAP, the ASCM and the Chattering Index, as discussed in the next three sections.

4.3. The HDAP

According to what has been described in section *2.2.4.3*, the creation of the High Density Alarm Plot develops through three steps. The first step is the creation of the binary database, which has already been described in paragraph *4.2*. Thus, the second and third steps only will be described now:

2. *<u>time bins creation and alarm count</u>*
   the observation period (i.e. 09/09/2017 16:07:24 – 07/10/2017 14:25:41) is divided in ten-minutes-spaced time intervals (called "bins"). The result is a time vector; an example is presented in *Table 4.3*.

09/09/2017 16:10:00
09/09/2017 16:20:00
09/09/2017 16:30:00
…
07/10/2017 14:10:00
07/10/2017 14:20:00
07/10/2017 14:30:00

Table 4.3 - Ten-minutes-spaced time vector

   Each element of the vector (e.g. 09/09/2017 16:20:00, a "bin") represents a 10 minutes time interval (e.g. the bin "09/09/2017 16:20:00" represents the time interval between 16:15:00 and 16:25:00 of the same day).
   For each time interval, and for each unique alarm, the alarm count within the time interval is calculated using the binary database as a data source. The result is stored in a vector of the same dimension of the time vector. In this way, each unique alarm is associated with a vector whose elements represent the number of alarm occurrences in a time interval equal to 10 minutes.
   The vectors are grouped to form a matrix. Each column of the matrix represents a unique alarm and each row represents a 10 minutes time interval. Each element of the matrix is an integer representing how many times the unique alarm occurred during the 10 minutes time intervals;

3. *HDAP creation*

   the matrix obtained in step *2* is used to calculate the total alarm count for each unique alarm (i.e. how many times each unique alarm occurred within the study period). A ranking system has been built to sort the unique alarms based on the total alarm count (i.e. higher rank means higher alarm count). The ranking/sorting is necessary to ensure that alarms with higher alarm count will be displayed higher in the HDAP (alarms must be sorted in such a way that the total alarm count decreases from the top to the bottom of the plot).

   Finally, the plot is built; the y-axis represents all the unique alarms and the x-axis represents the ten minutes spaced time bins. A colour bar has been built to correctly colour-coding the points of the plot, as described in section *2.2.4.3*.

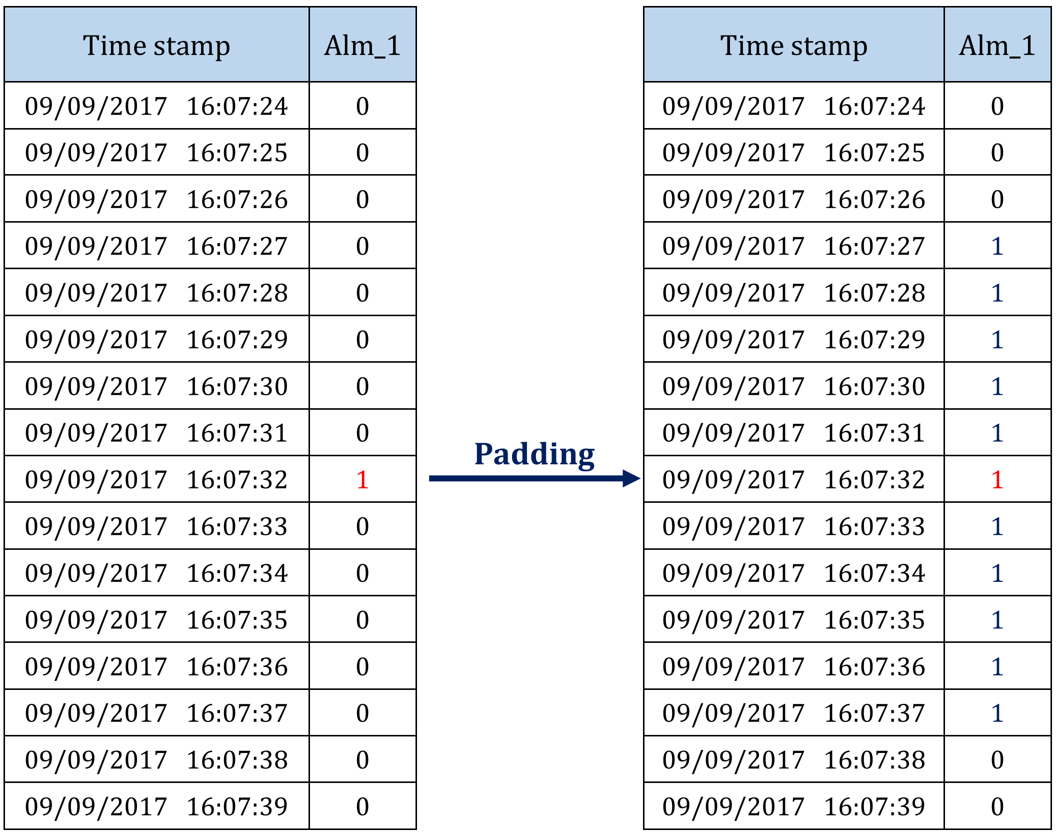
Through these steps, an HDAP containing 763 unique alarms (y-axis) and 1598 ten-minutes-spaced time bins (x-axis) is obtained. The plot is too large to be displayed correctly, for this reason, it has been decided to display a reduced version, containing the first 15 alarms only (i.e. the 15 alarms with the higher alarm count: the "top 15 bad actors"). The HDAP of the top 15 bad actors is presented in *Figure 5.1*.

4.4. The ASCM

According to what has been told in section *2.2.4.2*, the creation of the Alarm Similarity Color Matrix requires five steps. The first step is the creation of the binary database, which has already been described in paragraph *4.2*. Thus, the steps from the second to the fifth will be described now:

2. *padding*

   each binary sequence (i.e. each column of the binary database) must be "padded" with extra "1's" to enrich the alarm occurrences and to consider a possible communication lag (Kondaveeti *et al.*, 2010). The padding phase consists of adding a user-defined number of ones before and after each alarm occurrence in the binary sequence (i.e. every time a "1" is found in the binary sequence, a certain number of ones are added before and after the occurrence). In this thesis, a padding length equal to 5 has been chosen (Kondaveeti *et al.*, 2010). Thus, five ones are placed before and after every occurrence. The general procedure is outlined in *Figure 4.1*.



Time stamp	Alm_1
09/09/2017 16:07:24	0
09/09/2017 16:07:25	0
09/09/2017 16:07:26	0
09/09/2017 16:07:27	0
09/09/2017 16:07:28	0
09/09/2017 16:07:29	0
09/09/2017 16:07:30	0
09/09/2017 16:07:31	0
09/09/2017 16:07:32	1
09/09/2017 16:07:33	0
09/09/2017 16:07:34	0
09/09/2017 16:07:35	0
09/09/2017 16:07:36	0
09/09/2017 16:07:37	0
09/09/2017 16:07:38	0
09/09/2017 16:07:39	0

Time stamp	Alm_1
09/09/2017 16:07:24	0
09/09/2017 16:07:25	0
09/09/2017 16:07:26	0
09/09/2017 16:07:27	1
09/09/2017 16:07:28	1
09/09/2017 16:07:29	1
09/09/2017 16:07:30	1
09/09/2017 16:07:31	1
09/09/2017 16:07:32	1
09/09/2017 16:07:33	1
09/09/2017 16:07:34	1
09/09/2017 16:07:35	1
09/09/2017 16:07:36	1
09/09/2017 16:07:37	1
09/09/2017 16:07:38	0
09/09/2017 16:07:39	0

Figure 4.1 - Padding procedure

"Alm_1" in *Figure 4.1* represents a generic unique alarm.

It is worth noting that the padding length is an adjustable parameter and must be chosen according to the problem of concern (Kondaveeti *et al.*, 2010).

3. *calculation of the similarity measure*

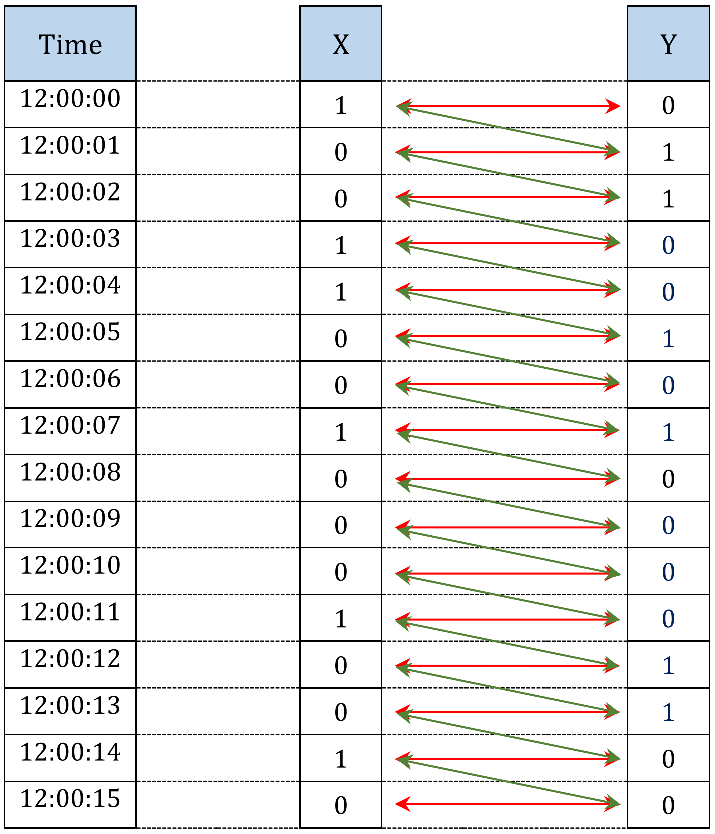
as previously argued, the Jaccard measure has been used in this thesis as a similarity indicator. The Jaccard measure is defined as (Kondaveeti *et al.*, 2010):

$$S_{Jacc}(X,Y) = \max_{l \in L}\left(\frac{a(l)}{a(l)+b(l)+c(l)}\right) \qquad 4.1$$

where:

- X and Y = two unique alarm's binary sequences (padded);
- L = a vector, containing the allowable delay times between the binary sequences;
- l = allowable time lag between the sequences;
- $a(l)$ = number of "matches" (i.e. $x_i = 1, y_i = 1$ being x_i and y_i elements of the sequences X and Y separated by a lag time equal to l);
- $b(l)$ = number of "mismatches" (i.e. $x_i = 1$, $y_i = 0$, being x_i and y_i elements of the sequences X and Y separated by a lag time equal to l);
- $c(l)$ = number of "mismatches" (i.e. $x_i = 0$, $y_i = 1$, being x_i and y_i elements of the sequences X and Y separated by a lag time equal to l).

It is worth noting that L is a vector whose elements are one-second-spaced integers representing the allowable time lags (in seconds). To clarify the use of the allowable time lag, an example is needed. For instance, let X and Y be the fictitious binary sequences presented in *Figure 4.2*:



Time	X	Y
12:00:00	1	0
12:00:01	0	1
12:00:02	0	1
12:00:03	1	0
12:00:04	1	0
12:00:05	0	1
12:00:06	0	0
12:00:07	1	1
12:00:08	0	0
12:00:09	0	0
12:00:10	0	0
12:00:11	1	0
12:00:12	0	1
12:00:13	0	1
12:00:14	1	0
12:00:15	0	0

Figure 4.2 - Similarity measure with time delay

Each element of the time sequences X and Y is one second spaced.

If $l = 0\ s$, the Jaccard measure is calculated with a time lag equal to 0 s. The calculation of "matches" and "mismatches" (equation 4.1) can be represented by the red arrows in *Figure 4.2*. In this situation, the matches and mismatches are:

- $a(l) = 1$;
- $b(l) = 5$;
- $c(l) = 5$.

Thus, for $l = 0\ s$, the similarity measure, according to equation 4.1, is:

$$S_{Jacc}(X,Y)_{l=0} = \frac{1}{1+5+5} = 0.091$$

On the other hand, if $l = 1\ s$, the Jaccard measure is calculated with a time lag equal to 1 s. The calculation of "matches" and "mismatches" (equation 4.1) can be represented by the green arrows in *Figure 4.2*. In this situation, the matches and mismatches are:

- $a(l) = 3$;
- $b(l) = 3$;
- $c(l) = 3$.

Thus, for $l = 1\ s$, the similarity measure, according to equation 4.1, is:

$$S_{Jacc}(X, Y)_{l=1} = \frac{3}{3 + 3 + 3} = 0.333$$

Therefore, in this example, the similarity is higher considering a lag time of 1 second.

The calculation must be repeated for each $l \in L$. Thus, for each pair of binary sequences, $\dim(L)$ similarity measures can be obtained (i.e. one measure for each element of L). According to equation 4.1, the "final" similarity measure between a couple of binary alarm data is the larger Jaccard measure.

In this thesis, an allowable lag (or lead) time of 4 minutes has been considered (Kondaveeti *et al.*, 2010). Hence, the vector L can be represent as:

$$L = [-240, -239, \dots, 0, \dots, 239, 240] \qquad 4.2$$

As previously argued, a total of 763 unique alarms have occurred within the study period. Thus, 291466 unique pairs of alarm binary sequences exist. The similarity measure is calculated for each of these pairs using the binary database as a source of data. Specifically, when a pair is selected, 481 Jaccard measures are calculated (one measure $\forall\ l \in L$ in equation 4.2) and, finally, the larger Jaccard measure is chosen to represent the similarity.

At the end of this step, a 763 x 763 matrix is obtained. The row and column indices represent a unique alarm and each element of the matrix is a Jaccard measure. Initially, the matrix is a lower unitriangular matrix (sub-diagonal elements only have been calculated since $S_{Jacc}(X, Y)_{l \in L} = S_{Jacc}(Y, X)_{l \in L}$). Finally, the matrix has been made symmetric.

At the end of this step, each alarm that occurred less than six times within the observation period has been removed from the matrix. This has been done because the Jaccard measure provides erroneously high values for couples of alarms that have very few "1's" in the binary sequences.

As a result, a reduced 407 x 407 correlation matrix has been obtained.

4. *re-ordering*

   The symmetric matrix obtained at the end of step three encloses the Jaccard measures for each couple of unique alarms with more than 5 occurrences. However, the alarms (i.e. the row and column indices) are displayed in alphabetical order; although rational, this choice is totally arbitrary and does not group correlated alarms into clusters.

   A pre-made clustering algorithm has been used to sort the matrix and shows clusters of correlated alarms. Specifically, "dendrogram" and "linkage" functions of "scipy.cluster.hierarchy" python library have been used. For the "linkage" function, the "average" method has been chosen.

   The result is a sorted matrix (rows and columns order has been rearranged) in which correlated alarms are displayed together according to the selected linkage method.

5. *colour coding*

   A convenient way to display the correlation matrix is to colour each element according to the Jaccard measure. A pre-made algorithm has been used to achieve this purpose. Specifically, the "heatmap" function of the "Seaborn" python library has been used. The "Reds" colourmap has been used; in this way, the colour of the elements fades from "brick red" (if the similarity measure is large -i.e. close to 1) to "light salmon" (if the similarity measure is low -i.e. close to 0).

The result of these steps is the ASCM. The matrix is too large to be displayed properly ($407 \, x \, 407$). For this reason, a reduced version of the ASCM (containing 70 unique alarms only) is presented in *Figure 5.3*.

4.5. The Chattering Index (ψ)

According to what has been told in section *2.2.4.1,* the calculation of the Chattering Index requires five steps. The first step is the creation of the binary database, which has already been described in paragraph *4.2*. Thus, the steps from the second to the fifth will be described now:

4. *Run-Length (r) calculation*

   a Run-Length is the "*time difference in seconds between two consecutive alarms on the same tag*" (Kondaveeti *et al.*, 2013). Thus, whenever a "1" is found in the binary representation of the alarm, the Run-Length is the time between the "1" and the next "1" in the binary sequence. A graphical representation of the Run-Lengths of fictitious alarm data is provided in *Table 4.4* and *Figure 4.3*.

S. No.	Alarm time stamp	Time count	Time difference (run length, *r*)
1	4/24/2010 12:00:01	1	3
2	4/24/2010 12:00:04	4	3
3	4/24/2010 12:00:07	7	5
4	4/24/2010 12:00:12	12	7
5	4/24/2010 12:00:19	19	7
6	4/24/2010 12:00:26	26	7
7	4/24/2010 12:00:33	33	2
8	4/24/2010 12:00:35	35	5
9	4/24/2010 12:00:40	40	7
10	4/24/2010 12:00:47	47	15
11	4/24/2010 12:01:02	62	–

Table 4.4 - Run length for a fictitious alarm based on historical data (Kondaveeti *et al.*, 2013)

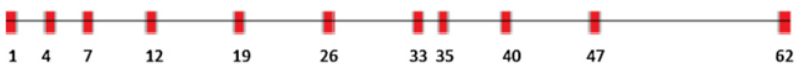


Figure 4.3 - Time trend showing alarm annunciations and the respective time count (Kondaveeti et al., 2013)

In *Table 4.4* the "Alarm time stamp" column represents the moments when the alarm was activated ("1" in the binary sequence). The "Time difference" column represents the Run-Lengths. For example, for the first alarm occurrence ("S. No." = 1), the Run-Length is 3 seconds; this is because between the first ad the second alarm occurrences there are 3 seconds. *Figure 4.3* is a time representation of the alarm occurrences.

Using the binary database as a data source, the Run-Lengths have been calculated for each unique alarm binary sequence.

5. <u>*Run length distribution (RLD) calculation*</u>

   intuitively, if most of the alarm occurrences have very short Run-Length, the alarm will show chattering behaviour. To quantify this insight, Kondaveeti *et al.* (2013) proposed to count how many times each Run-Length (e.g. 3 seconds) is repeated in the alarm binary sequence (e.g. Run-Length 3 s recurs 2 times in *Table 4.4*). The results can be displayed in a histogram (*Figure 4.4*).

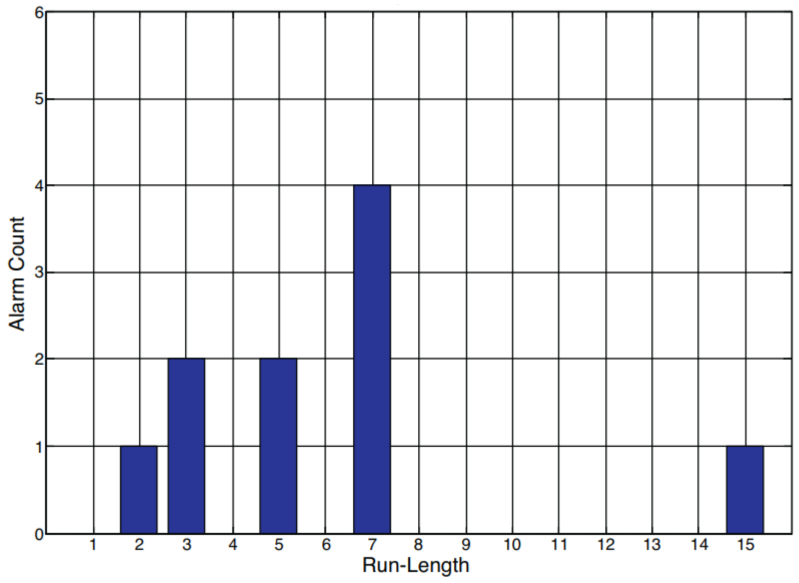


Figure 4.4 - Run length distribution for the fictitious alarm presented in Figure 4.3 and Table 4.4 (Kondaveeti *et al.*, 2013)

   Figure 4.4 represents the Run-Length Distribution of the alarm data presented in *Table 4.4*. The x-axis represents the Run-Length value (r), the y-axis represents how many times an alarm with a specific Run-Length has occurred (n_r).

   For each unique alarm in the binary database, the Alarm Counts associated with the Run-Lengths (obtained in step 4) have been calculated.

   The result is a list of Run-Lengths and associated Alarm Counts; each unique alarm has its own list.

6. <u>*Discrete Probability Function (DPF) calculation*</u>;

   Run-Length Distribution data can be normalized to obtain the Discrete Probability Function. Normalizing with a factor $\sum_r n_r$ lead to:

$$P_r = \frac{n_r}{\sum_r n_r} \qquad \forall\ r \in \mathbb{N} \tag{4.3}$$

where:

- $r =$ a Run-Length;
- $n_r =$ the Alarm Count associated with the Run-Length r;
- P_r represents the probability that the run length r occurred n_r times.

Thus, for each unique alarm, if the alarm is associated with z uniques Run-Lengths, z Discrete Probability Functions are calculated.

7. *Chattering index (ψ) calculation*.
   the chattering index of a unique alarm can be calculated by summing the products between each probability function (P_r) and the reciprocal of the run length (r):

$$\psi = \sum_{r \in \mathbb{N}} P_r \frac{1}{r} \tag{4.4}$$

The reciprocal of the run length is used as a weighting function to emphasize the alarm count with short run lengths (Kondaveeti *et al.*, 2013).
As a result, a Chattering index is calculated for each unique alarm. If the Chattering index is larger than a threshold value (e.g. 0.05 alarms/s, see equation 2.3), the alarm will be labelled as a chattering one.

The steps described above produce a single Chattering index for each unique alarm. Thus, observing the index, one can assess whether the alarm showed chattering or not. Although meaningful, this is a static result (i.e. if, for instance, the alarm showed chattering behaviour for one day only, the chattering index would conclude that the alarm was chattering, with no further specifications). Since the aim of this thesis is to dynamically assess chattering, a more flexible and dynamic tool is needed.
For this reason, The Dynamic chattering index has been developed. The ideas and assumptions that headed to this new index are described in the next section.

4.5.1. The Dynamic chattering index (ψ_D)

The purpose of this thesis is to develop a machine learning tool for real-time chattering assessment. Ideally, every time an alarm occurs, the algorithm should predict whether the alarm is going to show chattering behaviour or not. The machine learning algorithm needs to be trained on a set of historical data. The aim of the Dynamic chattering index is to associate each alarm occurrence (i.e. "1" in the binary sequence) with a measure that quantifies the tendency of the alarm to show chattering in the future. The Dynamic chattering index represents the raw data that, after some manipulations, are fed to the machine learning algorithm during the training phase.
The core idea behind the Dynamic chattering index is to calculate a "regular" Chattering index every time a "1" is found in the binary sequence. This way, for each unique alarm, as many chattering indices as alarm occurrences are calculated (in opposition to the original chattering index, which is just one for each unique alarm).
As previously argued, when an alarm occurs, we want to define whether the alarm will show chattering *in the future*. But, "*the future*" must be defined. In this thesis it has the meaning of "*within one hour after the occurrence"*; this is a design parameter and it can be adjusted. Considering a "short" future time (i.e. few minutes) could be a problem, because the Chattering index is a statistical tool; the more the available data, the more the index is reliable. On the other hand, considering a far future would cause a loss in dynamism (i.e. the farther the future considered, the more the Dynamic chattering index is similar to the regular Chattering index).
The algebra behind the Dynamic chattering index is the same as the original one. For this reason, an example is provided to clarify how the Dynamic Chattering index is obtained, and to highlight the differences with the original one.

Let Z be the binary sequence associated with the fictitious alarm "Alm_1", as displayed in *Table 4.5*.

Index	Time stamp	Z
1	09/09/2017 16:07:24	1
2	09/09/2017 16:09:25	1
3	09/09/2017 16:09:30	1
4	09/09/2017 16:09:33	1
5	09/09/2017 16:09:47	1
6	09/09/2017 16:09:59	1
7	09/09/2017 16:10:57	1
8	09/09/2017 16:15:00	1
9	09/09/2017 16:50:24	1
10	09/09/2017 17:05:42	1
11	09/09/2017 17:50:15	1
...	...	1
i	$Time\ stamp_i$	1
...	...	1
n	$Time\ stamp_n$	1

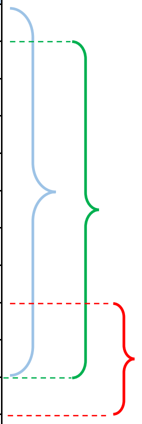


Table 4.5 - Binary sequence of the fictitious alarm "Alm_1". Parentheses represent the reduced binary sequences for the Dynamic chattering index calculation

The “Index” column in *Table 4.5* has been added to support the future explanation. For visualization purposes, all the zeroes have been removed from the binary sequence (i.e. column “Z”, the binary sequence, contains 1’s only).

The calculation of the Dynamic chattering indices is iterative and, being n the total number of occurrences (*Table 4.5*), it proceeds through $n-1$ iteration. Each generic iteration i comprises the following steps:

1. the occurrence with Index = i is selected (e.g. Index = 1 for the first iteration);
2. part of the binary sequence is now selected. Specifically, only the occurrences happened *within 1 hour* after the occurrence with Index = i are selected (e.g. the blue graph parenthesis for the first iteration, the green one for the second, the red one for the ninth, and so on);
3. using the reduced binary sequence obtained during step 2, a “regular” Chattering index is calculated, as it has been already described at the beginning of paragraph *4.5*. The index is stored in a vector of the same dimension of the binary sequence. Specifically, it is stored in position i.

Finally, the steps from 1 to 3 are repeated $\forall\ i \in \mathbb{N},\ i < n$.

This way, every time a “1” is found in the binary sequence, a chattering index is calculated, which represents the tendency of the alarm to show chattering within one hour after the occurrence. This chattering index is called the Dynamic chattering index (ψ_D), since it dynamically assesses chattering every time an alarm occurs.

The vector containing the Dynamic chattering indices can be grouped together with the binary sequence as shown in *Table 4.6*.

Index	Time stamp	Z	ψ_D Alm_1
1	09/09/2017 16:07:24	1	0.08
2	09/09/2017 16:09:25	1	0.079
3	09/09/2017 16:09:30	1	0.057
4	09/09/2017 16:09:33	1	0.019
5	09/09/2017 16:09:47	1	0.012
6	09/09/2017 16:09:59	1	0.002
7	09/09/2017 16:10:57	1	0.0006
8	09/09/2017 16:15:00	1	0.0001
9	09/09/2017 16:50:24	1	0.0001
10	09/09/2017 17:05:42	1	...
11	09/09/2017 17:50:15	1	...
...	...	1	...
i	$Time\ stamp_i$	1	...
...	...	...	...
n	$Time\ stamp_n$	1	\

Table 4.6 - Binary sequence and Dynamic chattering indices of the fictitious alarm "Alm_1"

The column "ψ_D Alm_1" in *Table 4.6* represents the vector of the Dynamic chattering indices associated with each alarm occurrence.
Similarly to the original Chattering index, a threshold value of 0.05 alarms/s can be used. But, in this case, the meaning of a Dynamic chattering index being higher than the threshold value is that the alarm will show chattering behaviour *within one hour* after the current occurrence (e.g. the red values in the last column of *Table 4.6*).

As a result, for each unique alarm, a vector containing the Dynamic chattering index has been obtained.

4.5.1.1. Limitations

The Chattering index is mathematically structured to perform better on large datasets. In the Dynamic chattering index calculation, only a relatively short (1 hour) binary sequence is used. For this reason, if very few alarms were triggered within one hour, the dynamic index might behave in an unexpected way. As an example, consider the binary sequence (Y) represented in *Table 4.7.*

Time stamp	Y	$r\ [s]$
09/09/2017 16:07:24	1	1
09/09/2017 16:07:25	1	32
09/09/2017 16:07:57	1	37
09/09/2017 16:08:34	1	1559
09/09/2017 16:34:33	1	\

Table 4.7 - Fictitious binary sequence (Y) and associated Run-Lengths (r)

Table 4.7 describes an alarm that occurred five times only within one hour. The last column of the table (r) represents the run lengths (as described at the beginning of paragraph *4.5)*.

The alarm itself does not suggest chattering (it occurred five times only) but, if one calculates the Dynamic chattering index of the first occurrence:

$$r = [1, 32, 37, 1559] \qquad 4.5$$

$$n_r = [1, 1, 1, 1] \qquad 4.6$$

Then, the DPF (equation 4.3) is:

$$P_r = [{}^1\!/_4, {}^1\!/_4, {}^1\!/_4, {}^1\!/_4] \qquad 4.7$$

Finally, the Dynamic chattering index related to the first occurrence is:

$$\begin{aligned} \psi_D &= {}^1\!/_4 \cdot 1 + {}^1\!/_4 \cdot {}^1\!/_{32} + {}^1\!/_4 \cdot {}^1\!/_{37} + {}^1\!/_4 \cdot {}^1\!/_{1559} = \\ &= 0.25 + 0.008 + 0.007 + 0.0002 = 0.265 > 0.05 \end{aligned} \qquad 4.8$$

Thus, according to the index, the alarm has shown chattering. This is because of the presence of an extremely short run length (i.e. 1 s) and a large P_r (i.e. 1/4). The contribution of the run length equal to 1 s would be enough to label the alarm as a chattering one (red value in equation 4.8).

To conclude, the proposed Dynamic chattering index is sensitive to “high probability–short run lengths” combinations and, in certain circumstances, it may behave in an unexpected way.

4.6. Tensor Flow simulations

The first step in developing a machine learning algorithm is to build a database containing both the *features* and the *labels* (section *2.3.1*). The selection of the most relevant features relies mainly on experience; a trial and error approach is frequently needed. Next, the database is split into two parts; the first part is used to train the model, the second part is used to evaluate it. Later, the model is selected (i.e. Linear, Deep or Wide&Deep), and the features are converted to fit the model needs. Finally, the model is trained and evaluated.
A more detailed description of the steps introduced above is now presented.

1. *database creation*

   data (i.e. *features* and *labels*) must be grouped and organized to form a database (i.e. a matrix). This is the most efficient way to prepare the data to be fed into the Machine Learning models. The database is organized in such a way that the features are columns of the database, and the labels are stored in the last column of the database. Each row of the database represents a list of alarm features and the related label (i.e. a row of the database represents a single alarm). The general structure of a hypothetic database is displayed in *Table 4.8*.

	Feature 1	Feature 2	...	Feature n	Label
Alarm_1	FI306	LLL	...	Feature_n_1	1
Alarm_2	LI315	HHH	...	Feature_n_2	0
...	...	...	...	...	...
Alarm_k	Feature_1_k	Feature_2_k	...	Feature_n_k	Label_k

Table 4.8 - Machine Learning database: general structure

A series of k alarm records, represented by n features, are shown in *Table 4.8.* In this example, the "Feature 1" is the alarm source (*Table 3.2*), the second feature is the alarm identifier (*Table 3.2*) and the labels are either "0" or "1" (i.e. a binary classification). It is worth noting that the alarms represented in the database (i.e. the rows) are not the "unique alarms" discussed in the previous sections; actually, they are alarm occurrences as represented in the original alarm database (section *3.3*).

As previously mentioned, selecting the most meaningful features often requires a trial and error approach. Several tests have been performed, only the features that have been used in the final simulations are now presented:

- the *year* when the alarm occurred (Y);
- the *month* when the alarm occurred (M);
- the *day* when the alarm occurred (D);
- the *hour* when the alarm occurred (H);
- the *minute* when the alarm occurred (m)
- the *second* when the alarm occurred (S);
- alarm *source* (SO)*;*
- alarm *identifier* (ID);
- alarm *condition name* (CN)*;*
- alarm *safety function* (JX);
- the *Active Time* (ATD);
- the *data value* (VAL);
- the *Eng. Units* (UNI).

All the attributes presented in the list above have been already discussed in (*Table 3.2*), except the *condition name*, which is the alarm *identifier* of the original alarm from the same source. For instance, an "HHH" alarm has "HHH" as *condition name*, but an "HHH Recovery" one has again "HHH" as *condition name*. This is because the alarm that has been recovered was an "HHH" one.
The time features (i.e. year, month, day etc.) have been derived by dividing the "Time Stamp" attribute. This has been done because Machine Learning models accept only numerical values as input. Certainly, strings can be used as well (i.e. words and phrases), but they must be "mapped" into numeric values before being fed into the model. For more about the mapping functions, see TensorFlow.org (TensorFlow.org, 2020c). For the moment, it is enough to know that if the Time Stamps were fed as string objects (e.g. "09/09/2017 16:07:24") the model would not be able to assess the connection between "near" events (e.g. "09/09/2017 16:07:24" and "09/09/2017 16:07:25" would have been considered two completely unrelated events). Representing the time in distinct features resolves the issue.

According to the abbreviation presented in the list above, *Table C. 5* shows a reduced version of the final database.

It is worth noting that the features have been entirely derived from the original alarm database. Thus, the required data are directly provided by DCS and no further manipulation is needed.

The last column of *Table C. 5* represents the *labels*; a label can be either zero or one. If the alarm is going to show chattering behaviour within one hour, the label is "1". In contrast, the label is "0", if the alarm is not going to show chattering. The binary representation of the labels has been derived from the Dynamic chattering index described in section *4.5.1*. For an alarm's occurrence, if the calculated Dynamic chattering index is greater than 0.05, a "1" is chosen as a label. On the contrary, a "0" is chosen as a label, if the related Dynamic chattering index is lower than 0.05.

The final database has 25572 rows (i.e. alarm occurrences within the observation period, section *3.3.1*) and 14 columns (13 features + 1 label).

2. *<u>training and evaluation databases</u>*

   the database is split into two parts. The first part will be used during the training phase, the second part will be used to evaluate the performance (i.e. the prediction capability) of the trained models. The number of available data is limited; thus, it has been decided to enrich the training database with more alarm data. The training database comprises ¾ of the original database. The remaining part constitutes the evaluation database.

   In the original database, the data (i.e. rows) are displayed in chronological order. This may cause problems when the database is split. For instance, it may happen that most of the chattering alarm "occurred" in the first part of the database (i.e. the training part). Similarly, it may happen that certain alarms are included in the evaluation database while they did not occur in the training database.

   Machine Learning algorithms prefer well-distributed data. For this reason, before splitting the original database, the rows have been randomly sorted (i.e. shuffle). This resolves poor data distribution.

In addition, tests have been performed using the non-shuffled database as well, to assess how the models will perform with poorly distributed data and unseen events.

Finally, the last column of the evaluation databases (i.e. the *labels*) has been removed and stored in a separate variable. This has been done because the aim of the evaluation phase is labels prediction; thus, the algorithm must not have access to the true labels' values during that phase.

3. *model selection and features conversion*

   three different models have been trained and evaluated (i.e. Linear, Deep, Wide&Deep). The models have been trained and evaluated using the same databases and the same features. Still, the features need to be converted based on the selected model. How the features are converted is out of the scope of this work, for more about feature conversion see TensorFlow.org (2020c) and the code in sections *B.1*, *B.2* and *B.3*.

   Table 4.9 summarize the features (e.g. numerical, categorical and crossed) used to train each model.

	Linear	Deep	Wide&Deep[1]
Numerical	"Y", "M", "D", "H", "m", "S", "ATD", "VAL".	"Y", "M", "D", "H", "m", "S", "ATD", "VAL"	"Y", "M", "D", "H", "m", "S", "ATD", "VAL".
Categorical	"SO", "ID", "CN", "JX", "UNI".	"SO", "ID", "CN", "JX", "UNI".	"SO", "ID", "CN", "JX", "UNI".
Crossed	"VAL" x "UNI" "SO" x "ID" x "CN" "SO" x "ID" x "JX"	\	"VAL" x "UNI" "SO" x "ID" x "CN" "SO" x "ID" x "JX".

Table 4.9 - Features summary

Categorical features are non-numerical features (e.g. strings). Crossed features have already been discussed in sections *2.3.3.1* and *2.3.3.3*.

[1] The Deep part uses Numerical and Categorical features, the Wide part uses Crossed features only.

To conclude, the Deep model and the deep part of the Wide&Deep model have three hidden layers. The first hidden layer has 1024 hidden units, the second 512 and the third 256.

4. <u>*training*</u>

   the models are trained using the training database as a data source. During this phase, the weights of the models are optimized to provide an accurate mapping from the input (the features) to the output (the labels). The Linear model and the linear part of the Wide&Deep model use the FTRL optimizer. The Deep model and the deep part of the Wide&Deep model use the "Adagrad" optimizer. All the models have been trained for the same number of steps.

5. <u>*evaluation*</u>

   the trained models are now evaluated against their ability to predict the correct labels. Each row of the evaluation database (i.e. the features related to a specific alarm occurrence) is fed into the model. As a result, the model provides the predicted label. Specifically, it predicts the label's probability (as discussed in section *2.3.1*). By default, the software uses a threshold equal to 0.5 to decide the predicted label's value. By comparing the predicted labels with the real labels, the software calculates and displays a collection of performance metrics (e.g. accuracy, recall, precision etc.). Although useful, these metrics are limited to the default threshold. For this reason, the raw labels' probability data have been manipulated to calculate precisions and recalls using different thresholds values. This has led to more meaningful tools to assess the performance of the models compared to the "default" performance metrics.

Chapter 5

Results

5.1. HDAP

In the present paragraph, the results of the process described in section *4.3* are presented. Specifically, *Figure 5.1* displays the HDAP of the top 15 bad actors over the whole observation period (defined in section *4.2*).

The observation period is large, the resulting plot (*Figure 5.1*) may appear a little zoomed-out; although readable, the points of the plot are thin, and it may be difficult to focus on a single bin. For this reason, a zoomed version of the plot is presented in *Figure 5.2*. In this figure, the same data are presented but the observation period has been reduced, and only the top 10 bad actors are considered (i.e. two days of observation are shown, from 09/09/2017 to 10/09/2017). The plot in *Figure 5.2* has been created to display a better definition and separation between the points (i.e. sticks) of the HDAP. The result is a less general, but more readable, plot compared to *Figure 5.1*.

The colour coding is displayed in the colour bar to the right of each figure. Grey sticks are a single alarm occurrence within the 10 minutes interval. From 2 to 10 occurrences the colour fades from green to orange. For occurrences larger than 10 the colour fades from orange to red. It is worth noting that the time presented under the x-axis of *Figure 5.1* and *Figure 5.2* is GMT time.

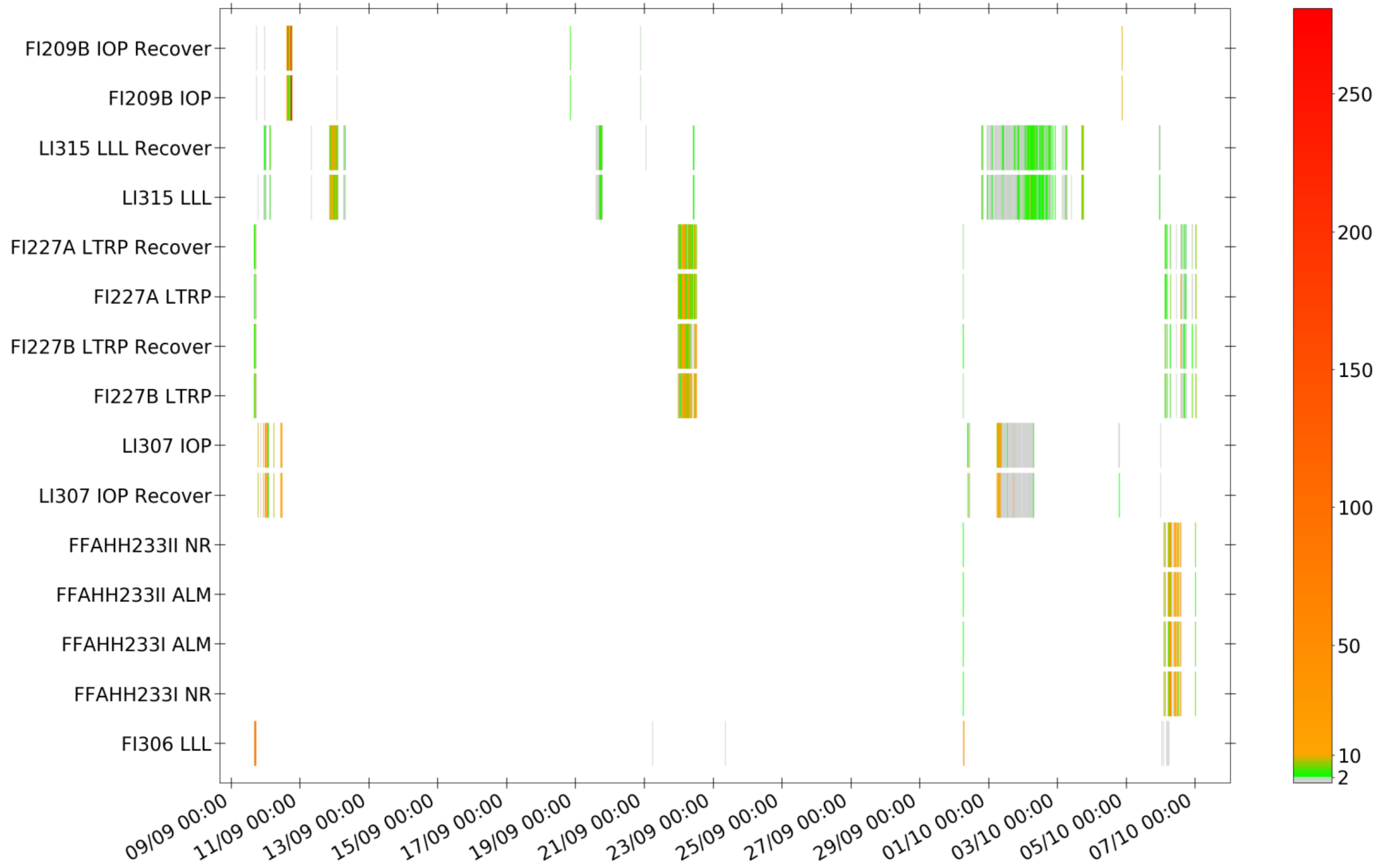


Figure 5.1 - HDAP of the Top 15 bad actors over the whole observation period

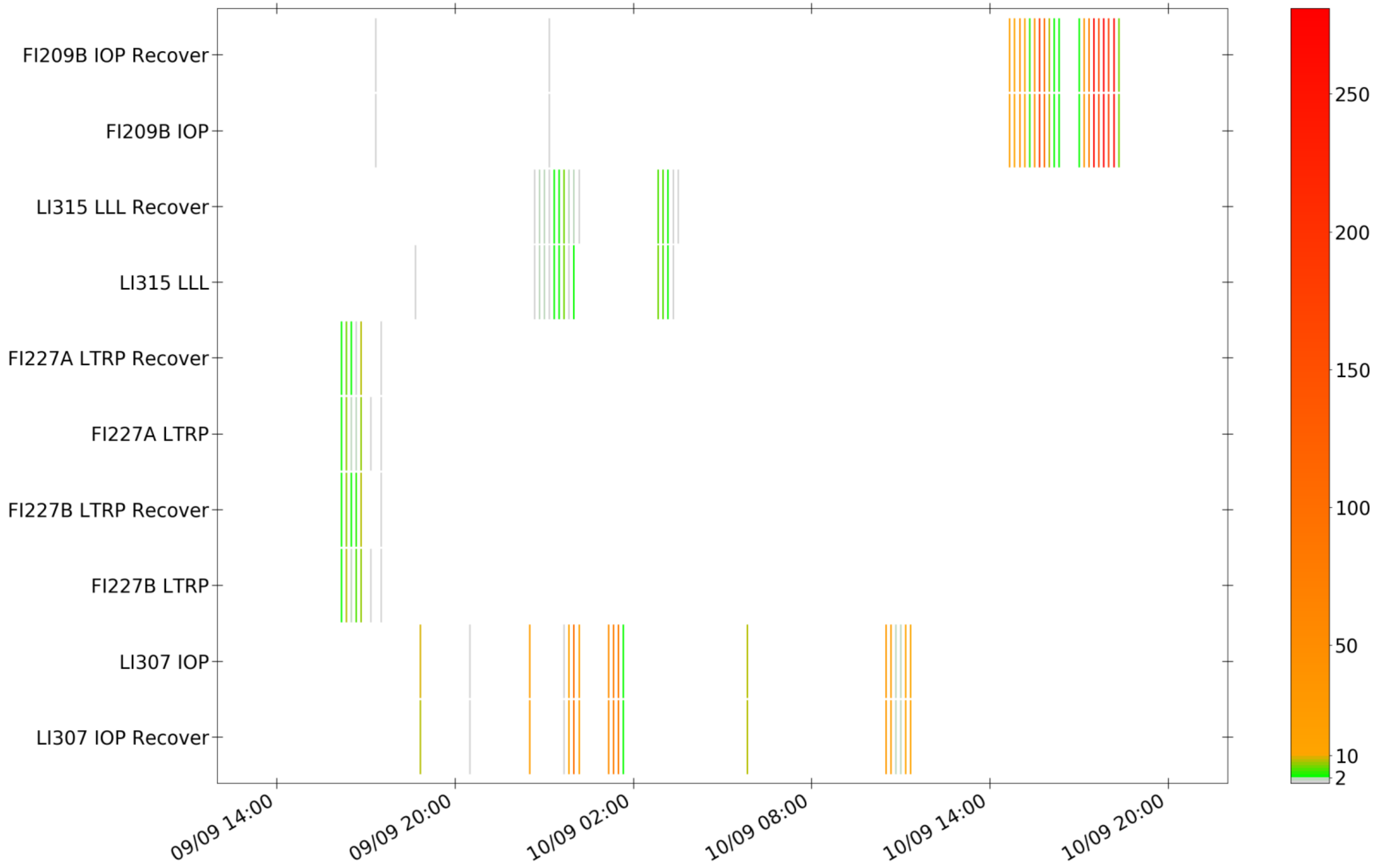


Figure 5.2 - Reduced version of the HDAP presented in Figure 5.1 (Top10 bad actors and reduced observation period)

Observing *Figure 5.1*, it is evident that the plant went through at least two instability periods. The first happened on the 9th of September, and it refers to the power outage. The second period of instability happened on the 30th of September.
Observing *Figure 5.2*, one can detect several alarms with very high alarm count (i.e. intense red "sticks" in the plot, e.g. FI209B, LI307). It is very likely that these alarms showed chattering behaviour during the period of concern (more than 100 alarms in 10 minutes is a strong chattering indication). In *Figure 5.2*, FI227A LTRP and FI227B LTRP show the same alarm distribution (same "sticks" position and colour); thus, it is highly probable that these two alarms are redundant (as suggested by their names).

5.2. ASCM

In the present paragraph, the result of the process described in section *4.4* is presented. Specifically, *Figure 5.3* displays a reduced version of the ASCM; it contains 70 elements only. The colour coding is displayed in the colour bar to the right of the plot; it reflects the value of the Jaccard measure of each couple of unique alarms.

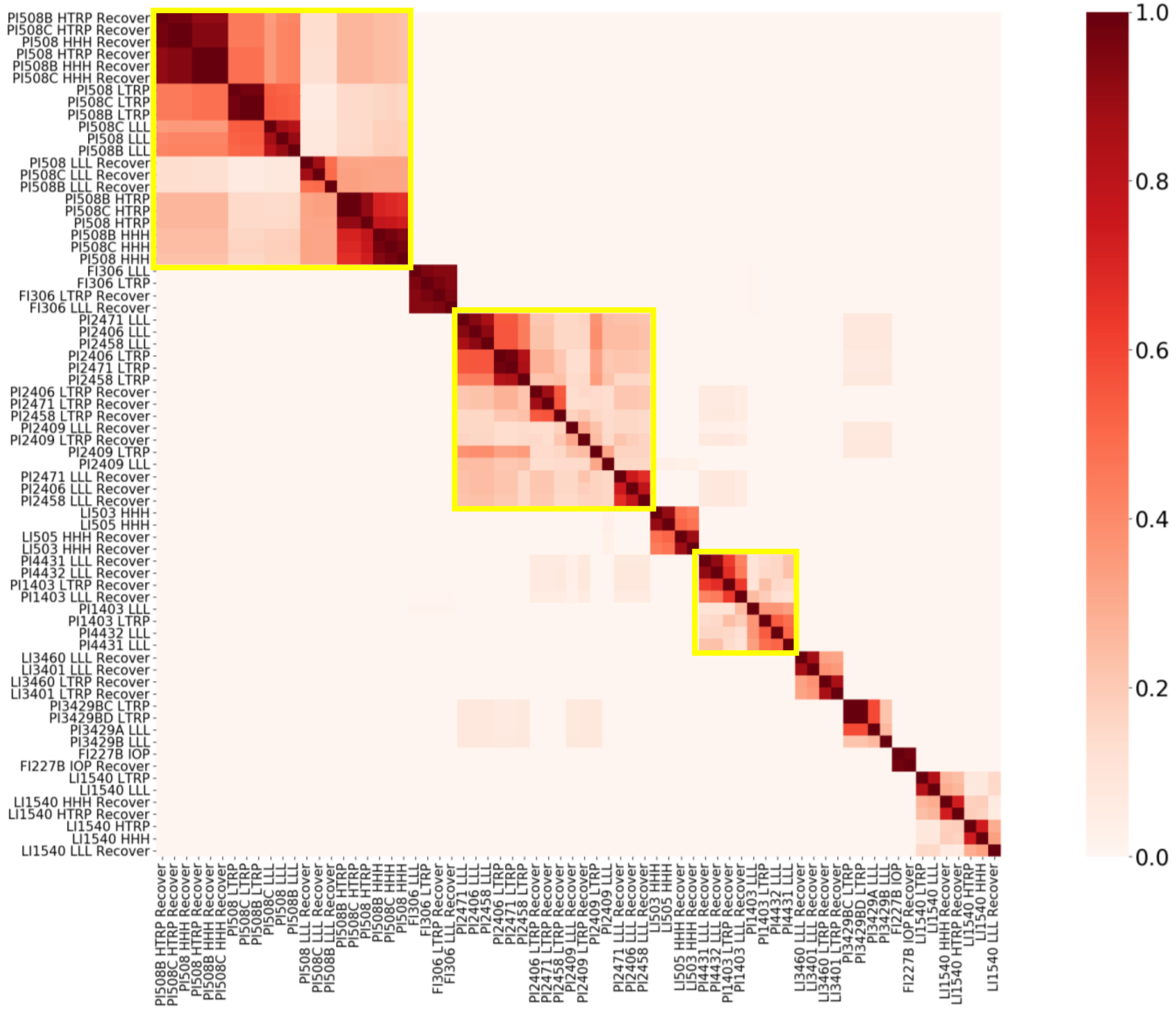


Figure 5.3 - Reduce ASCM (70 unique alarms only)

Observing *Figure 5.3*, several clusters can be identified. For example, three clusters are highlighted by a yellow border.
From left to right, the first highlighted cluster suggests that P508, PI508B and PI508C are highly correlated and, maybe, redundant.
The second cluster suggests that PI2471, PI2406 and PI2458 are highly correlated. Furthermore, some kind of correlation exists between those alarms and PI2409.
The third cluster highlights a strong correlation between PI4431 and PI4432. Furthermore, some kind of correlation exists between those alarms and PI1403.

5.3. Chattering index

Since the aim of this thesis is the dynamic assessment of chattering, the process described at the beginning of paragraph *4.5* has been tested on two unique alarms only. This has been done to ensure that the python code written to calculate the "regular" Chattering index works properly, because the calculation of the Dynamic chattering indices involves the calculation of a "regular" chattering index on a reduced binary sequence (point *3* of section *4.5.1*).

The Chattering index has been calculated for two unique alarms:

a. FI209B IOP;
b. LI318 LTRP Recover.

"FI209B IOP" is the first of the bad actors (*Figure 5.1*) and shows apparent chattering. The latter is out of the list of the top 20 bad actors (not displayed in this thesis) and it never occurred more than two times within a 10 minutes time interval; thus, it is apparently not chattering.
The RLD plots of the two alarms are displayed in *Figure 5.4* and *Figure 5.5*.

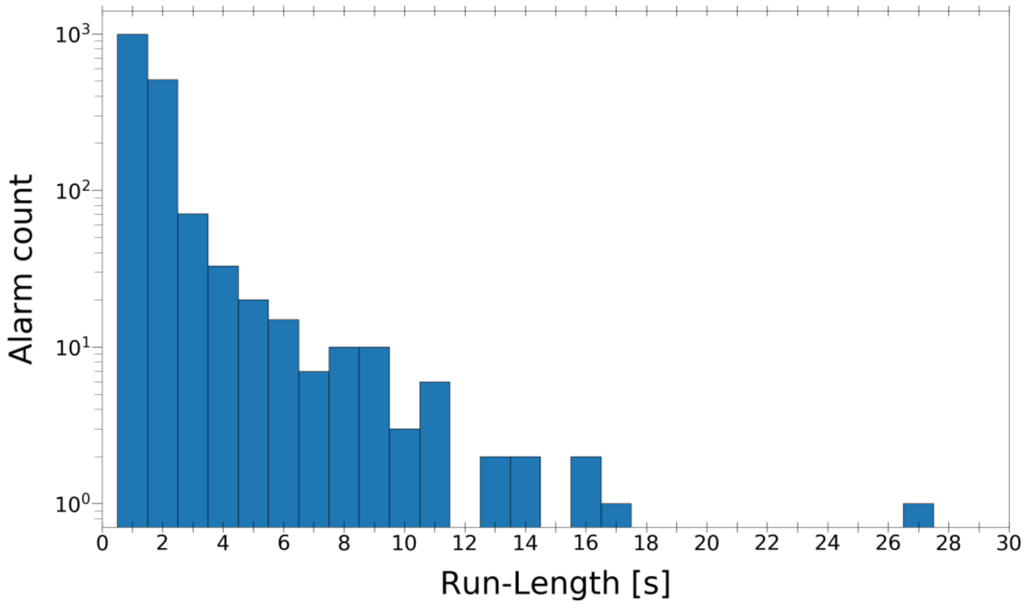


Figure 5.4 - RLD histogram of FI209B IOP

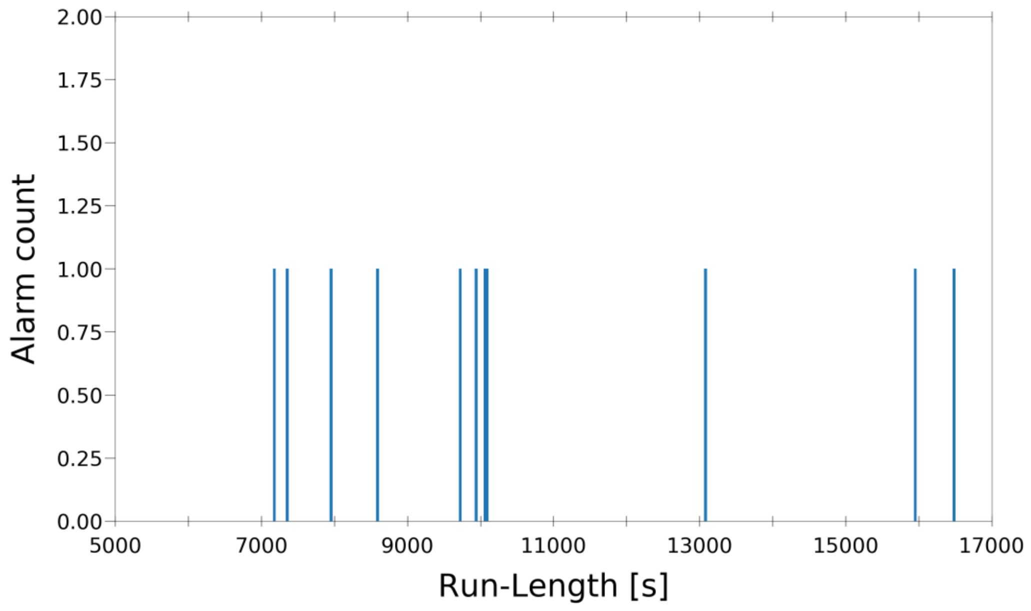


Figure 5.5 - RLD histogram of LI318 LTRP Recover

Figure 5.4 shows a peak (high Alarm count) for short Run-Lengths (i.e. 1 to 10 seconds), this is an evident indicator of chattering. Oppositely, *Figure 5.5* presents no peaks; the Alarm count is low (i.e. 1) and the Run-Lengths are large (i.e. > 7000 s); together, this information indicates a non-chattering alarm.
Finally, the Chattering index has been calculated:

a. $\psi_{FI209B\ IOP} = 0.748 > 0.05\ alarms/s\ \rightarrow chatteering$;
b. $\psi_{LI318\ LTRP\ Recover} = 7.73 \cdot 10^{-5} < 0.05\ alarms/s\ \rightarrow not\ chatteering$.

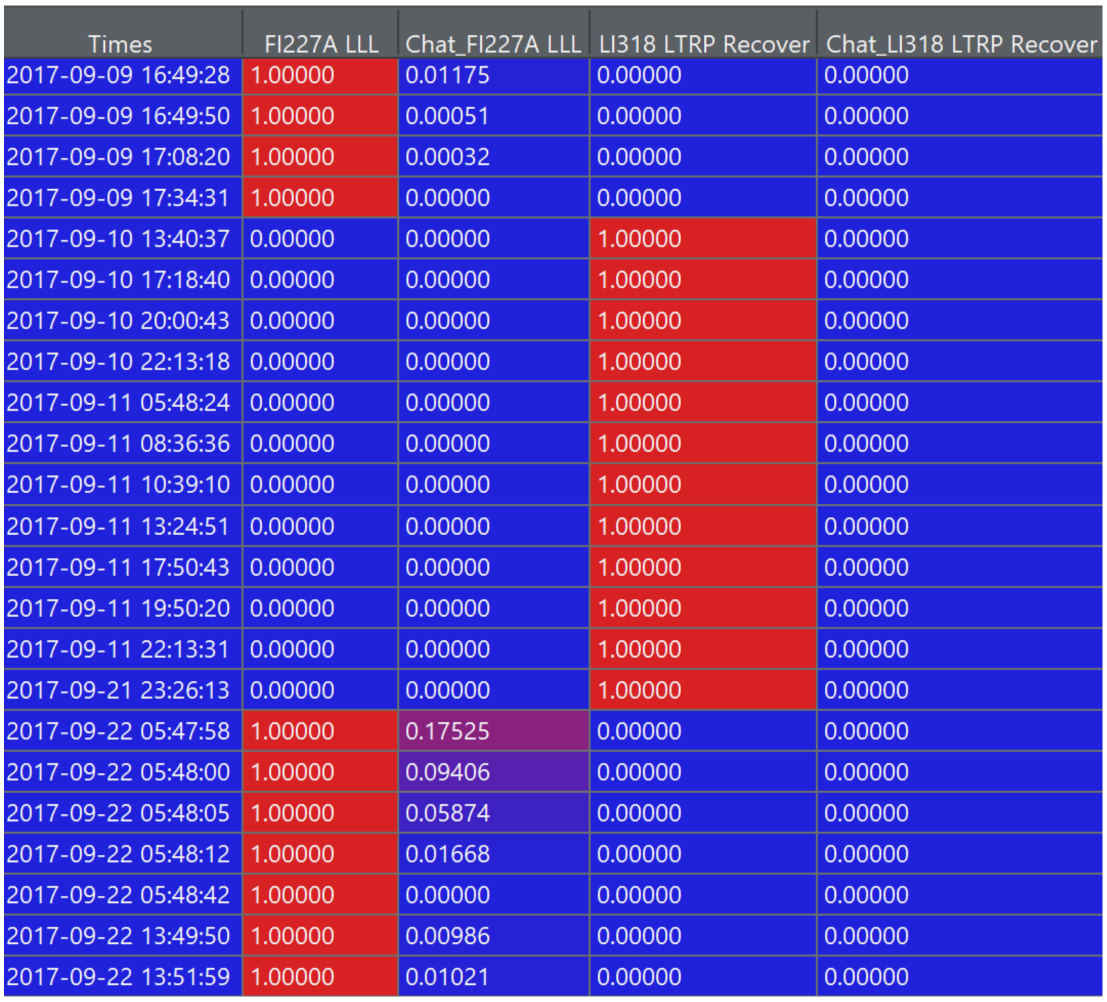
The results are coherent with what has been told so far about the two alarms.

5.3.1. Dynamic chattering index

A small portion of the results obtained from the process described in section *4.5.1* are shown in *Table 5.1*. Specifically, the figure is a screenshot of the IDE (Integrated Development Environment) that has been used to code the python scripts. Two unique alarms are displayed in the figure:

a. FI227A LLL;
b. LI318 LTRP Recover.

The second and the fourth columns of *Table 5.1* are the binary sequences of “FI227A LLL” and “LI318 LTRP Recover” respectively. The third and last columns contain the related Dynamic chattering indices. The first column represents the time stamp.



Times	FI227A LLL	Chat_FI227A LLL	LI318 LTRP Recover	Chat_LI318 LTRP Recover
2017-09-09 16:49:28	1.00000	0.01175	0.00000	0.00000
2017-09-09 16:49:50	1.00000	0.00051	0.00000	0.00000
2017-09-09 17:08:20	1.00000	0.00032	0.00000	0.00000
2017-09-09 17:34:31	1.00000	0.00000	0.00000	0.00000
2017-09-10 13:40:37	0.00000	0.00000	1.00000	0.00000
2017-09-10 17:18:40	0.00000	0.00000	1.00000	0.00000
2017-09-10 20:00:43	0.00000	0.00000	1.00000	0.00000
2017-09-10 22:13:18	0.00000	0.00000	1.00000	0.00000
2017-09-11 05:48:24	0.00000	0.00000	1.00000	0.00000
2017-09-11 08:36:36	0.00000	0.00000	1.00000	0.00000
2017-09-11 10:39:10	0.00000	0.00000	1.00000	0.00000
2017-09-11 13:24:51	0.00000	0.00000	1.00000	0.00000
2017-09-11 17:50:43	0.00000	0.00000	1.00000	0.00000
2017-09-11 19:50:20	0.00000	0.00000	1.00000	0.00000
2017-09-11 22:13:31	0.00000	0.00000	1.00000	0.00000
2017-09-21 23:26:13	0.00000	0.00000	1.00000	0.00000
2017-09-22 05:47:58	1.00000	0.17525	0.00000	0.00000
2017-09-22 05:48:00	1.00000	0.09406	0.00000	0.00000
2017-09-22 05:48:05	1.00000	0.05874	0.00000	0.00000
2017-09-22 05:48:12	1.00000	0.01668	0.00000	0.00000
2017-09-22 05:48:42	1.00000	0.00000	0.00000	0.00000
2017-09-22 13:49:50	1.00000	0.00986	0.00000	0.00000
2017-09-22 13:51:59	1.00000	0.01021	0.00000	0.00000

Table 5.1 – Dynamic chattering indices of "FI227A LLL" and "LI318 LTRP Recover" (reduced observation period)

5.4. Tensor Flow simulations

In the present paragraph, the results of the TensorFlow simulations are presented. As previously argued, two batches of simulations have been performed (section *4.6*, step *2*). In the first simulations, the database was shuffled before being split. In the second simulations, it was not shuffled.

5.4.1. First simulations (shuffled)

The main results of the first simulations are now presented. The evaluation phase led to the metrics presented in *Table 5.2*.

	Accuracy	Precision	Recall
Linear	0.947	0.941	0.938
Deep	0.937	0.929	0.926
Wide&Deep	0.919	0.919	0.892

Table 5.2 – Accuracy, Precision and Recall of the first simulations

The confusion matrices linked to the models are displayed in *Figure 5.6*.

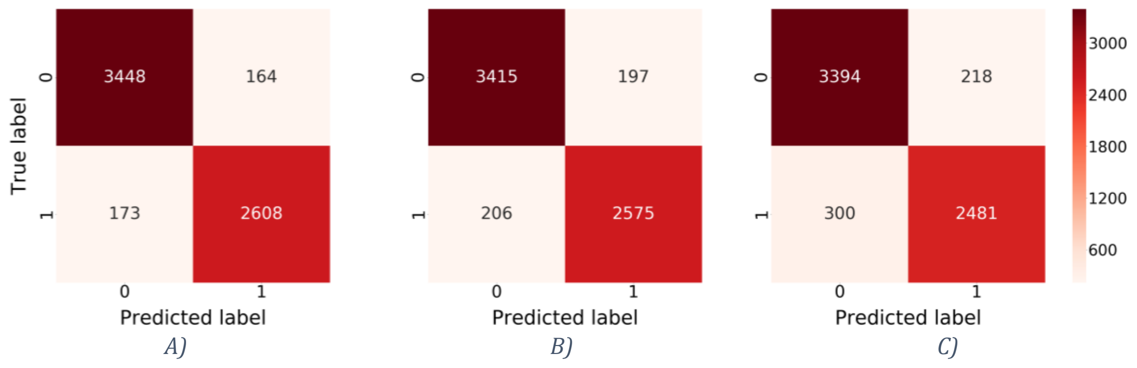


Figure 5.6 - Confusion Matrices of the first simulations.
A) = Linear model, B) = Deep model, C) = Wide&Deep model

It is worth noting that the results presented above refer to a "default" threshold equal to 0.5. As an example, the Precision-Recall curve related to the Linear model is displayed in *Figure 5.7*.

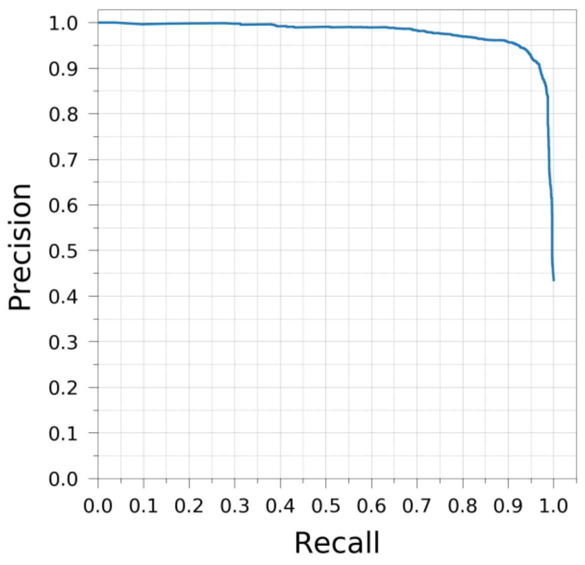


Figure 5.7 - Precision-Recall curve of the Linear model (first simulations)

The Precision-Recall curves generated by the Deep and Wide&Deep models are similar to the one presented in *Figure 5.7.* Thus, they are not displayed in this thesis.

Observing the metrics in *Table 5.2,* one can conclude that the Linear model produces larger metrics. Similarly, the Deep model produces larger metrics than the Wide&Deep does. Studying the confusion matrices displayed in *Figure 5.6,* it is evident that the models are weaker when it comes to predicting the label "1"; the number of False Negatives (bottom left of the confusion matrices) is always higher than the number of False Positives (top right of the confusion matrices).

5.4.1. Second simulations (non-shuffled)

The main results of the second simulations are now presented. The evaluation phase led to the metrics presented in *Table 5.3*.

	Accuracy	Precision	Recall
Linear	0.502	0.635	0.021
Deep	0.492	0.309	0.009
Wide&Deep	0.492	0.2	0.004

Table 5.3 - Accuracy, Precision and Recall of the second simulations

The confusion matrices linked to the models are displayed in *Figure 5.8*.

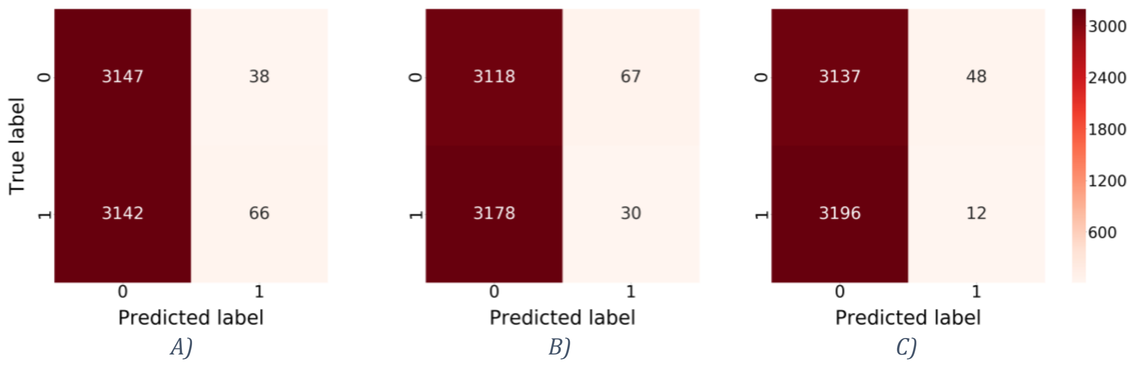


Figure 5.8 - Confusion Matrices of the second simulations.
A) = Linear model, B) = Deep model, C) = Wide&Deep model

It is worth noting that the results presented above refer to a “default” threshold equal to 0.5.

Observing *Table 5.3*, one can conclude that the metrics follow the same trend observed in the first simulations (i.e. the Linear model produces larger metrics than the other models, the Deep model produces larger metrics than the Wide&Deep one). Similarly, studying the confusion matrices displayed in *Figure 5.8*, one can observe the same trend as in the first simulations. But here, the trend is exaggerated, with more than 3000 False Negatives. Comparing *Table 5.2* and *Figure 5.6* with *Table 5.3* and *Figure 5.8*, it is evident that the models performed worst on the non-shuffled database.

The results of the second simulations have been studied more in-depth; the threshold has been varied to assess whether better metrics could be obtained. The Precision-Recall curves related to the models are presented in *Figure 5.9*, *Figure 5.10*, *Figure 5.11*.

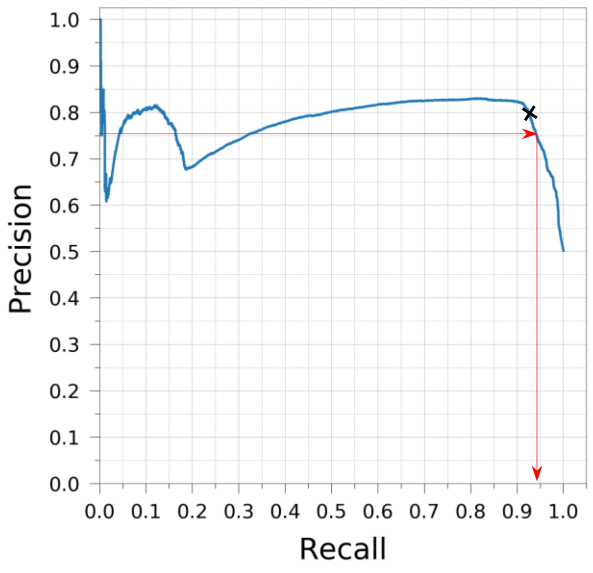


Figure 5.9 - Precision-Recall curve of the Linear model (second simulations)

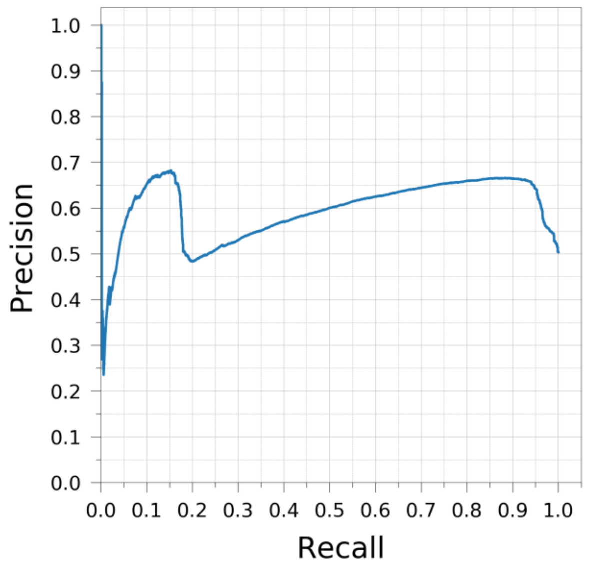


Figure 5.10 - Precision-Recall curve of the Deep model (second simulations)

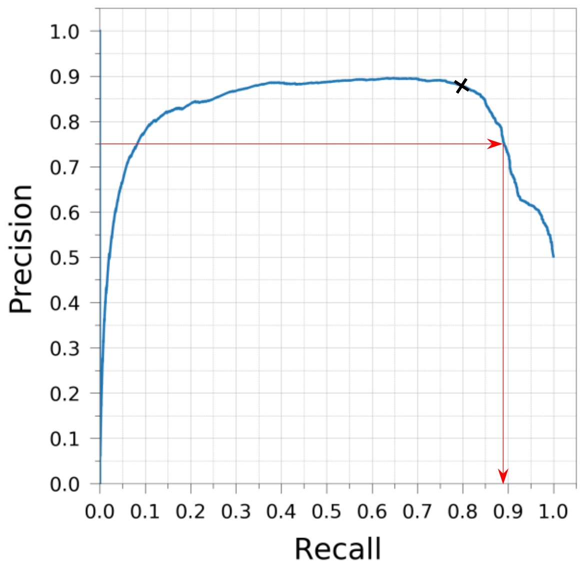


Figure 5.11 - Precision-Recall curve of the Wide&Deep model (second simulations)

The shapes of the Precision-Recall curves displayed in *Figure 5.9* and *Figure 5.10* are similar. But the linear model can achieve higher precisions and recalls compared to the Deep model. Furthermore, the Wide&Deep model (*Figure 5.11*) can achieve higher precision compared to the Linear model.

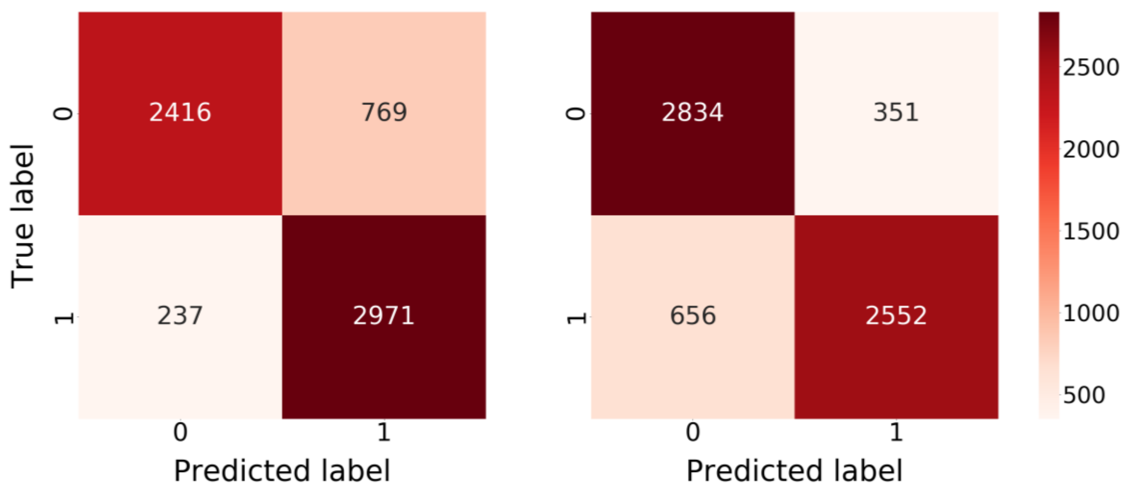
If a Precision accessible to both the Linear and the Wide&Deep models is selected, the Recall provided by the Wide&Deep model will be lower. For instance, red arrows in *Figure 5.9* and *Figure 5.11* highlight the Recall for a Precision equal to 0.75; in these conditions, the Linear model reaches a Recall greater than 0.9 while the Wide&Deep model cannot reach 0.9.

As an example, two points of the Precision-Recall curves related to the Linear and Wide&Deep models have been further analysed. The points are displayed as black crosses in *Figure 5.9* and *Figure 5.11*. The metrics related to these points, and the linked thresholds, are displayed in *Table 5.4*.

	Threshold	Precision	Recall
Linear	0.0001	0.794	0.926
Wide&Deep	0.068	0.879	0.795

Table 5.4 - Thresholds, Precision and Recall for the points marked in Figure 5.9 and Figure 5.11

The confusion matrices related to the points under assessment are displayed in *Figure 5.12*.



A) *B)*

Figure 5.12 - Confusion Matrices of the points marked in Figure 5.9 and Figure 5.11

A) = Linear model, B) =Wide& Deep model

Observing the two matrices above, one can conclude that the Linear model is better in predicting the label “0”. On the contrary, the Wide &Deep model better predicts the “1” label.

Chapter 6
Discussion

6.1. Introduction

The results presented in *Chapter 5* are now evaluated and discussed. The limitations of the analyses are highlighted, as well as the difficulties experienced during the application of the methods. Finally, recommendations for further works are provided.

6.2. HDAP, ASCM, Chattering index

Through the HDAP (*Figure 5.1* and *Figure 5.2*), periods of plant instability are highlighted, and bad actors are revealed. Furthermore, the HDAP can be used for preliminary chattering and redundancy assessment. A large HDAP (i.e. based on a wide study period) is useful to assess instability and bad actors, while chattering and redundancy assessment is easier on a reduced version of the plot.
Correlation between alarms can be studied through the ASCM. Sometimes the matrix displays trivial correlations (e.g. P508, PI508B and PI508C) but interesting correlations are displayed as well (e.g. PI2471, PI2406 and PI2458). When handling large alarm databases, it is convenient to "split" the ASCM into smaller, more readable,

matrices. The ASCM is consistent with the preliminary redundancy assessment performed observing the HDAP.
The Chattering index has been only used to test the algorithm; however, the results are consistent with the preliminary chattering assessment. The Dynamic chattering index permits to evaluate the likelihood that an alarm will show chattering behaviour within a defined time interval. As previously stated (section *4.5.1*), the method may behave unexpectedly when few alarms occurred within the time interval and when alarms with "high probability–short run lengths" are studied.

It is worth noting that these techniques require an impressive computational effort. For example, producing the binary database required ten days of uninterrupted computing. Similarly, the ASCM required sixteen days of uninterrupted computing. Furthermore, the computers used during the analyses were connected to the NTNU IT infrastructure; thus, they occasionally shut down during the simulations. The computational aspect must not be underestimated when approaching these techniques. The use of a reliable working station or, better, a supercomputer, would significantly simplify the analyses.

To conclude, it is confirmed that the techniques discussed above are valuable offline analysis tools. They can be used to support the Monitoring and assessment phase of the Alarm management lifecycle (*Figure 2.5*), and to improve the alarm system performances. For example, when chattering is identified, the alarm attributes (e.g. deadband, setpoint, delays) may be changed to resolve the issue. Similarly, when the ASCM detects redundancy, the priority level of one of the redundant alarms may be lowered.

Future research should examine the effect of different padding lengths when building the ASCM. In addition, the method for the calculation of the Dynamic chattering index needs to be improved to increase its reliability. Finally, using a larger database could significantly improve the quality of the results of the techniques discussed above.

6.3. TensorFlow simulations.

6.3.1. First simulations

The models show remarkable performances when the shuffled database is used (*Table 5.2*). Specifically, the Linear model provides the highest metrics, and, in this example, it qualifies as the best model. The reasons why a simpler model can overcome more modern and complex models may include:

1. the problem may require more memorization and fewer generalization capabilities;
2. the Deep and Wide&Deep models may overgeneralize, and detect a relationship between features where no (or poor) relationship exist;
3. the chosen set of features may not be suitable for Deep models;
4. Deep and Wide&Deep models are more sensitive and difficult to train.

It is worth noting that the models have been trained on the same set of features, and no effort have been invested in optimizing the models. Hundreds of different configurations exist for Deep and Deep&Wide models (e.g. different number of hidden units, different optimizers, learning decay, target metrics etc.). Due to time constraints as well as knowledge constraints, it has been decided to use “default” settings only. This may work properly for “robust” models, like the Linear one, but optimization may be needed for more sensitive and advanced models.

6.3.2. Second simulations

When the non-shuffled database is used, the models reveal weak performance metrics (i.e. metrics based on threshold = 0.5, *Table 5.3*). The reasons why this happens may include:

1. alarm data are not well spread among the two databases (i.e. training and evaluation);
2. the evaluation database may comprise alarms or situations that never occurred in the training database.

Nevertheless, if the threshold is varied, decent performances are achieved (*Table 5.4*). Specifically, the Precision-Recall curves of the Linear and the Wide&Deep models underline a significative potential for performance improvement (*Figure 5.9* and *Figure 5.11*). However, the required thresholds are relatively low (i.e. 0.0001 for the Linear, 0.068 for the Wide&Deep); this may be interpreted as a general weakness of the model in predicting the label "1" (i.e. the model appears to be confident in predicting the "0" labels, while the "1" labels are usually linked to low probability levels). The threshold related to the Linear model seems unrealistically low (i.e. 0.1%) while the one related to the Wide&Deep is low but it may be acceptable (6.8 %).
In predicting alarm chatter, the most important metric is believed to be the Precision. This is because predicting False Negatives is less critical than predicting False Positives. For instance, if a non-chattering alarm is wrongly labelled as chattering (i.e. a False Positive), actions may be taken to silence the alarm; in this way, legit alarms may be missed.
In this case, the model that optimizes the Precision is the Wide&Deep (*Figure 5.11* and *Table 5.4*). Thus, the Wide&Deep model may qualify as the best model when the non-shuffled database is used. The reason why this happens may include:

1. generalization capability is needed to assess unseen events or feature combinations (events or alarms that occurred in the evaluation database only) (Cheng *et al.*, 2016). Similarly, memorization capability is needed to extract knowledge from frequently occurring events or feature combinations (Cheng *et al.*, 2016).
2. the Deep model has excellent generalization capabilities but lacks in memorization;
3. the Linear model strongly relies on memorization but lacks in generalization.

In this context (i.e. non-shuffled database, poorly distributed and unseen events), the joint training of a Deep part and a Linear part may lead to better performances.

To conclude, the proposed Machine learning models achieved considerable performance in predicting chattering alarms. Furthermore, the required computational effort is negligible compared to the techniques described in the previous section (i.e. a prediction can be obtained in a fraction of a second). The method appears promising, and may be used for on-line, real-time, alarm assessment and classification. The results of the models (i.e. predictions) may be used in advanced alarming techniques (section *2.2.1.2*) to support the operator decision-making procedure.

Future research should certainly further test whether extending the analysis on a larger database will solve the issue of poor data distribution. Additionally, investigating the fields of “Feature Engineering” and “Data Mining” is an interesting topic for the future, since this may allow finding a more meaningful and convenient set of features. Similarly, optimizing the Deep and Wide&Deep models may lead to better performances. Finally, a Machine Learning model may be developed for real-time correlation prediction. As a long-term objective, future research should be devoted to the development of a method to integrate the Machine Learning models on a real industrial alarm system.

Chapter 7
Conclusions

A Machine Learning method for chattering prediction has been developed. The method has involved the creation of an index for dynamic chattering assessment. Three models have been trained and evaluated on a real industrial database; Linear, Deep and Wide&Deep models. The performances of the models have been evaluated against the ability to predict chattering alarms. In general, the models have shown good prediction capabilities. Chattering alarms and alarm floods are major drawbacks in modern alarm systems; the models proposed in the present work are flexible and dynamic tools that may be used to address these issues. The obtained Machine Learning models may be used for real-time chattering prediction; the results may be used to develop advanced alarming techniques. This would significantly improve the performance of the alarm systems and the operator response.

In addition, advanced alarm management techniques have been performed on the same alarm database. The results of the analyses have confirmed that these models are valuable offline tools to periodically monitor and assess the alarm system. The results of these techniques are meaningful, but static. Furthermore, a considerable computational effort is required.

To conclude, the two approaches are not in contrast; on the contrary, they complement each other. The advanced alarm management techniques are solid and reliable tools for offline, periodic, evaluations. Instead, the strength of a Machine Learning approach lies in the ability to predict future events. The first approach may be used on a regular basis, to maintain the alarm system; the second one may be used as an on-line tool, to improve the overall system flexibility and the operator response during critical events, like alarm floods.

Appendix A
Acronyms

ASCM	Alarm Similarity Color Matrix
BFW	Boiling Feed Water
BPCS	Basic Process Control System
DEA	Diethanolamine
DPF	Discrete probability function
FMEA	Failure Modes and Effects Analysis
HAZOP	Hazard and Operability study
HDAP	High Density Alarm Plot
IDE	Integrated Development Environment
IoT	Internet of Things
HMI	Human-Machine Interface
HPS	High Pressure Steam
LPS	Low Pressure Steam
MPS	Medium Pressure Steam

P&ID	Piping and Instrumentation Diagram
PFD	Process Flow Diagram
PLC	Programmable Logic Controller
RLD	Run Length Distribution
SIS	Safety Instrumented System

Appendix B

Code

B.1 Linear Model

```
from __future__ import absolute_import, division, print_function,
unicode_literals
import numpy as np
import pandas as pd
import seaborn as sns
from IPython.display import clear_output
from sklearn.metrics import roc_curve
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_recall_curve
from matplotlib import pyplot as plt
import tensorflow as tf
from matplotlib.ticker import (MultipleLocator)

tf.enable_eager_execution()

# Path to train database
train_file = '<path_to_training_file>'

# Path to evaluation database
test_file = '<path_to_evaluation_file>'

# Open the databases
df_train = pd.read_csv(train_file)
df_test = pd.read_csv(test_file)

# Drop columns containing chattering indices (it must not be passed as a
feature)
df_train = df_train.drop(['Chattering_Indices'], axis=1)
df_test = df_test.drop(['Chattering_Indices'], axis=1)
```

```
# Define the names of the columns of the two databases
COLUMNS = ["TSTR", "Y", "M", "D", "H", "m", "S", "TCOU", "TSTA", "TSNS", "SO",
           "ID", "TID", "CN", "JX", "ATSTR", "ATCOU", "ATSTA", "ATD", "VAL",
           "UNI", "CHD", "CHB"]

df_train.columns = COLUMNS
df_test.columns = COLUMNS

# Copy the Labels in different variables and drop them from the original
database
y_train = df_train.pop('CHB')
y_eval = df_test.pop('CHB')

# Define Numerical and Categorical columns
CATEGORICAL_COLUMNS = ['SO', 'ID', 'CN', 'JX', 'UNI']
NUMERIC_COLUMNS = ['Y', 'M', 'D', 'H', 'm', 'S', 'ATD', 'VAL']

# Drop unnecessary columns from the database
features = CATEGORICAL_COLUMNS + NUMERIC_COLUMNS
for element in COLUMNS[0:-1]:
    if element not in features:
        df_train.drop(columns=element, inplace=True)
        df_test.drop(columns=element, inplace=True)

# Convert the features and store them in a list
feature_columns = []
for feature_name in CATEGORICAL_COLUMNS:
    vocabulary = df_train[feature_name].unique()
    feature_columns.append(
    tf.feature_column.categorical_column_with_vocabulary_list(feature_name,
    vocabulary))

for feature_name in NUMERIC_COLUMNS:
    feature_columns.append(tf.feature_column.numeric_column(feature_name,
    dtype=tf.float64))

# Define the Crossed features
crossed = [tf.feature_column.crossed_column(["VAL", "UNI"],
hash_bucket_size=int(1e6)),
           tf.feature_column.crossed_column(["SO", "ID", "CN"],
hash_bucket_size=int(1e6)),
           tf.feature_column.crossed_column(["SO", "ID", "JX"],
hash_bucket_size=int(1e6))]


# Input pipeline function (from DataFrame to DataSet)
def make_input_fn(data_df, label_df, num_epochs=10, shuffle=True,
batch_size=32):
    def input_function():
        ds = tf.data.Dataset.from_tensor_slices((dict(data_df), label_df))
        if shuffle:
            ds = ds.shuffle(1000)
        ds = ds.batch(batch_size).repeat(num_epochs)
        return ds

    return input_function
```

```python
# Define the training and evaluation DataSets
train_input_fn = make_input_fn(df_train, y_train)
eval_input_fn = make_input_fn(df_test, y_eval, num_epochs=1, shuffle=False)

# Build the linear Classifier (the model directory can be specified by
model_dir='path_to_modeldir')
linear_est = tf.estimator.LinearClassifier(feature_columns=feature_columns +
crossed)

# Train the Classifier
linear_est.train(train_input_fn)

# Evaluate the Classifier
result = linear_est.evaluate(eval_input_fn)

# Print the default metrics
clear_output()
print(result)

# Assess predictions on the evaluation database
predictions = linear_est.predict(eval_input_fn)  # this is a dict, the raw
prediction probabilities are stored here
expected = y_eval  # these are the real labels

# Create a DataFrame containing the predictions (threshold=0.5), the real
Labels, and the prediction probabilities
list_predict = []
list_proba = []
list_expect = []
for pred_dict, expec in zip(predictions, expected):  # fill the lists
    class_id = pred_dict['class_ids'][0]
    probability = pred_dict['probabilities'][class_id]
    list_predict.append(class_id)
    list_proba.append(probability * 100)
    list_expect.append(expec)

df_predictions = pd.DataFrame({'Predictions': list_predict, 'Expected':
list_expect, 'Probabilities': list_proba})

# Manually calculate Tp, Fn, Fp, Recall, Precision
tp = 0
fp = 0
fn = 0
tn = 0
for p, e in zip(df_predictions.Predictions, df_predictions.Expected):
    if e == 1 and p == 1:
        tp += 1
    elif p == 1 and p != e:
        fp += 1
    elif p == 0 and p != e:
        fn += 1
    elif p == 0 and e == 0:
        tn += 1

Recall = tp / (tp + fn)
Precision = tp / (tp + fp)
```

```
# Take the probabilities of a positive prediction (i.e. "1") and store them
into the DataFrame
predictions = linear_est.predict(eval_input_fn)
probs = pd.Series([pred['probabilities'][1] for pred in predictions])  # this
is the probability of the label being "1"
df_predictions['Prob_1'] = probs

# Save the  Dataframe as a .csv text file
df_predictions.to_csv('<path_to_save_directory>\\file_name.csv', index=False)

# Plot the ROC curve
fpr, tpr, _ = roc_curve(y_eval, probs)
plt.plot(fpr, tpr)
plt.title('ROC curve')
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.xlim(0, )
plt.ylim(0, )
plt.show()

# Obtain the confusion matrix
conf_matrix = confusion_matrix(df_predictions.Expected,
df_predictions.Predictions)
df_conf_matrix = pd.DataFrame(conf_matrix, index=[0, 1], columns=[0, 1])

# Plot the confusion matrix (parameters are optimized for A4, 3 matrices in a
row)
fig = plt.figure()
ax = fig.add_subplot(111)
fig = sns.heatmap(df_conf_matrix, annot=True, square=True, cmap='Reds',
fmt='d', annot_kws={"size": 60})
cax = plt.gcf().axes[-1]
cax.tick_params(labelsize=55)
ax.set_ylabel('True label', fontsize=70, labelpad=30)
ax.set_xlabel('Predicted label', fontsize=70, labelpad=30)
ax.tick_params(labelsize=60, axis='x')
ax.tick_params(labelsize=60, axis='y')
plt.show()

# Build precision - recall curve using a built-in function (parameters are
optimized for A4, 2 or more figures per page)
precisions_1, recalls_1, thresholds_1 =
precision_recall_curve(df_predictions.Expected, df_predictions.Prob_1)
fig_2 = plt.figure(figsize=(5, 5))
ax = fig.add_subplot(111)
ax.plot(recalls_1, precisions_1, linewidth=5)
ax.set_xlabel('Recall', fontsize=60, labelpad=30)
ax.set_xlim(0, )
ax.xaxis.set_major_locator(MultipleLocator(0.1))  # set ticks in x axis
ax.xaxis.set_minor_locator(MultipleLocator(0.05))  # set ticks in x axis
ax.set_ylim(0, )
ax.yaxis.set_major_locator(MultipleLocator(0.1))  # set ticks in x axis
ax.yaxis.set_minor_locator(MultipleLocator(0.05))  # set ticks in x axis
ax.set_ylabel('Precision', fontsize=60, labelpad=30)
ax.tick_params(which='major', direction='out', length=20, width=1, pad=20,
labelsize=40, axis='x')
ax.tick_params(which='major', direction='out', length=20, width=1, pad=20,
labelsize=40, axis='y')
ax.tick_params(which='minor', direction='out', length=10, width=1, pad=20,
axis='x')
ax.tick_params(which='minor', direction='out', length=10, width=1, pad=20,
axis='y')
ax.grid(which='major', axis='both', linestyle='-')
ax.grid(which='minor', axis='both', linestyle='--')
plt.gca().set_aspect('equal', adjustable='box')  # make the plot squared

# Build precision - recall curve manually
recalls = []
precisions = []
thresholds = list(np.arange(0, 1, 0.0001))  # a list of thresholds
for thresh in thresholds:
    list_prediction_threshold = []
    for element in df_predictions.Prob_1:
        if element >= thresh:
            list_prediction_threshold.append(1)
        else:
            list_prediction_threshold.append(0)
    conf = confusion_matrix(df_predictions.Expected,list_prediction_threshold)
    true_p = conf[1][1]
    false_p = conf[0][1]
    false_n = conf[1][0]
    recalls.append(true_p / (true_p + false_n))
    precisions.append(true_p / (true_p + false_p))

# Store results in a DataFrame and drop nan (when there are no more Tp)
df_thresholds = pd.DataFrame({'Recall': recalls, 'Precision': precisions,
'Threshold': thresholds})
df_thresholds.dropna(inplace=True)

# Save the DataFrame
df_thresholds.to_csv('<path_to_save_directory>\\file_name.csv', index=False)
```

B.2 Deep Model

```python
from __future__ import absolute_import, division, print_function,
unicode_literals
import numpy as np
import pandas as pd
import seaborn as sns
from IPython.display import clear_output
from sklearn.metrics import roc_curve
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_recall_curve
from matplotlib import pyplot as plt
import tensorflow as tf
from matplotlib.ticker import (MultipleLocator)

tf.enable_eager_execution()

# Path to train database
train_file = '<path_to_training_file>'

# Path to evaluation database
test_file = '<path_to_evaluation_file>'

# Open the databases
df_train = pd.read_csv(train_file)
df_test = pd.read_csv(test_file)

# Drop columns containing chattering indices (it must not be passed as a
feature)
df_train = df_train.drop(['Chattering_Indices'], axis=1)
df_test = df_test.drop(['Chattering_Indices'], axis=1)

# Define the names of the columns of the two databases
COLUMNS = ["TSTR", "Y", "M", "D", "H", "m", "S", "TCOU", "TSTA", "TSNS", "SO",
           "ID", "TID", "CN", "JX", "ATSTR", "ATCOU", "ATSTA", "ATD", "VAL",
           "UNI", "CHD", "CHB"]

df_train.columns = COLUMNS
df_test.columns = COLUMNS

# Copy the Labels in different variables and drop them from the original
database
y_train = df_train.pop('CHB')
y_eval = df_test.pop('CHB')

# Define Numerical and Categorical columns
CATEGORICAL_COLUMNS = ['SO', 'ID', 'CN', 'JX', 'UNI']
NUMERIC_COLUMNS = ['Y', 'M', 'D', 'H', 'm', 'S', 'ATD', 'VAL']

# Drop unnecessary columns from the database
features = CATEGORICAL_COLUMNS + NUMERIC_COLUMNS
for element in COLUMNS[0:-2]:
    if element not in features:
        df_train.drop(columns=element, inplace=True)
        df_test.drop(columns=element, inplace=True)
```

```
# Convert the features and store them in a list
feature_columns = []
categorical_feature = []  # categorical, must be mapped into dense
for feature_name in CATEGORICAL_COLUMNS:
    vocabulary = df_train[feature_name].unique()
    categorical_feature.append(
    tf.feature_column.categorical_column_with_vocabulary_list(feature_name,
    vocabulary))

# Convert (wrap) categorical columns into dense columns:
# Indicator
for cat_col in categorical_feature:
    feature_columns.append(tf.feature_column.indicator_column(cat_col))

# Or Embedding_column
'''
for cat_col in categorical_feature:
    feature_columns.append(tf.feature_column.embedding_column(cat_col,
dimension=9))
'''

# Add numeric features
for feature_name in NUMERIC_COLUMNS:
    feature_columns.append(tf.feature_column.numeric_column(feature_name,
    dtype=tf.float64))


# Input pipeline function (from DataFrame to DataSet)
def make_input_fn(data_df, label_df, num_epochs=10, shuffle=True,
batch_size=32):
    def input_function():
        ds = tf.data.Dataset.from_tensor_slices((dict(data_df), label_df))
        if shuffle:
            ds = ds.shuffle(1000)
        ds = ds.batch(batch_size).repeat(num_epochs)
        return ds

    return input_function


# Define the training and evaluation DataSets
train_input_fn = make_input_fn(df_train, y_train)
eval_input_fn = make_input_fn(df_test, y_eval, num_epochs=1, shuffle=False)

# Build the Deep Classifier (the model directory can be specified by
model_dir='<path_to_modeldir>')
classifier_deep = tf.estimator.DNNClassifier(feature_columns=feature_columns,
hidden_units=[1024, 512, 256])

# Train the Classifier
classifier_deep.train(train_input_fn)

# Evaluate the Classifier
results = classifier_deep.evaluate(eval_input_fn)

# Print the default metrics
clear_output()
print(results)
```

```
# Assess predictions on the evaluation database
predictions = classifier_deep.predict(eval_input_fn)  # this is a dict, the
raw prediction probabilities are stored here
expected = y_eval  # these are the real labels

# Create a DataFrame containing the predictions (threshold=0.5), the real
Labels, and the prediction probabilities
list_predict = []
list_proba = []
list_expect = []
for pred_dict, expec in zip(predictions, expected):
    class_id = pred_dict['class_ids'][0]
    probability = pred_dict['probabilities'][class_id]
    list_predict.append(class_id)
    list_proba.append(probability * 100)
    list_expect.append(expec)

df_predictions = pd.DataFrame({'Predictions': list_predict, 'Expected':
list_expect, 'Probabilities': list_proba})

# Manually calculate Tp, Fn, Fp, Recall, Precision etc.
tp = 0
fp = 0
fn = 0
tn = 0
for p, e in zip(df_predictions.Predictions, df_predictions.Expected):
    if e == 1 and p == 1:
        tp += 1
    elif p == 1 and p != e:
        fp += 1
    elif p == 0 and p != e:
        fn += 1
    elif p == 0 and e == 0:
        tn += 1

Recall = tp / (tp + fn)
Precision = tp / (tp + fp)

# Take the probabilities of a positive prediction (i.e. "1") and store it into
the DataFrame
predictions = classifier_deep.predict(eval_input_fn)
probs = pd.Series([pred['probabilities'][1] for pred in predictions])  # this
is the probability of the label being "1"
df_predictions['Prob_1'] = probs

df_predictions.to_csv('<path_to_save_directory>\\file_name.csv', index=False)

# Plot the ROC curve
fpr, tpr, _ = roc_curve(y_eval, probs)
plt.plot(fpr, tpr)
plt.title('ROC curve')
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.xlim(0, )
plt.ylim(0, )
plt.show()
```

```
# Obtain the confusion matrix
conf_matrix = confusion_matrix(df_predictions.Expected,
df_predictions.Predictions)
df_conf_matrix = pd.DataFrame(conf_matrix, index=[0, 1], columns=[0, 1])

# Plot the confusion matrix (parameters are optimized for A4, 3 matrices in a
row)
fig = plt.figure()
ax = fig.add_subplot(111)
fig = sns.heatmap(df_conf_matrix, annot=True, square=True, cmap='Reds',
fmt='d', annot_kws={"size": 60})
cax = plt.gcf().axes[-1]
cax.tick_params(labelsize=55)
ax.set_ylabel('True label', fontsize=70, labelpad=30)
ax.set_xlabel('Predicted label', fontsize=70, labelpad=30)
ax.tick_params(labelsize=60, axis='x')
ax.tick_params(labelsize=60, axis='y')
plt.show()

# Build precision - recall curve using a built-in function (parameters are
optimized for A4, 2 or more figures per page)
precisions_1, recalls_1, thresholds_1 =
precision_recall_curve(df_predictions.Expected, df_predictions.Prob_1)
fig_2 = plt.figure(figsize=(5, 5))
ax = fig.add_subplot(111)
ax.plot(recalls_1, precisions_1, linewidth=5)
ax.set_xlabel('Recall', fontsize=60, labelpad=30)
ax.set_xlim(0, )
ax.xaxis.set_major_locator(MultipleLocator(0.1))  # set ticks in x axis
ax.xaxis.set_minor_locator(MultipleLocator(0.05))  # set ticks in x axis
ax.set_ylim(0, )
ax.yaxis.set_major_locator(MultipleLocator(0.1))  # set ticks in x axis
ax.yaxis.set_minor_locator(MultipleLocator(0.05))  # set ticks in x axis
ax.set_ylabel('Precision', fontsize=60, labelpad=30)
ax.tick_params(which='major', direction='out', length=20, width=1, pad=20,
labelsize=40, axis='x')
ax.tick_params(which='major', direction='out', length=20, width=1, pad=20,
labelsize=40, axis='y')
ax.tick_params(which='minor', direction='out', length=10, width=1, pad=20,
axis='x')
ax.tick_params(which='minor', direction='out', length=10, width=1, pad=20,
axis='y')
ax.grid(which='major', axis='both', linestyle='-')
ax.grid(which='minor', axis='both', linestyle='--')
plt.gca().set_aspect('equal', adjustable='box')  # make the plot squared

# Build precision - recall curve manually
recalls = []
precisions = []
thresholds = list(np.arange(0, 1, 0.0001))  # a list of thresholds
for thresh in thresholds:
    list_prediction_threshold = []
    for element in df_predictions.Prob_1:
        if element >= thresh:
            list_prediction_threshold.append(1)
        else:
            list_prediction_threshold.append(0)
    conf = confusion_matrix(df_predictions.Expected,
list_prediction_threshold)
```

```
    true_p = conf[1][1]
    false_p = conf[0][1]
    false_n = conf[1][0]
    recalls.append(true_p / (true_p + false_n))
    precisions.append(true_p / (true_p + false_p))

# Store results in a DataFrame and drop nan (rise when there are no more Tp)
df_thresholds = pd.DataFrame({'Recall': recalls, 'Precision': precisions,
'Threshold': thresholds})
df_thresholds.dropna(inplace=True)

# Save the DataFrame
df_thresholds.to_csv('<path_to_save_directory>\\file_name.csv', index=False)
```

B.3 Wide&Deep Model

```python
from __future__ import absolute_import, division, print_function,
unicode_literals
import numpy as np
import pandas as pd
import seaborn as sns
from IPython.display import clear_output
from sklearn.metrics import roc_curve
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_recall_curve
from matplotlib import pyplot as plt
import tensorflow as tf
from matplotlib.ticker import (MultipleLocator)

tf.enable_eager_execution()

# Path to train database
train_file = '<path_to_training_file>'

# Path to evaluation database
test_file = '<path_to_evaluation_file>'

# Open the databases
df_train = pd.read_csv(train_file)
df_test = pd.read_csv(test_file)

# Drop columns containing chattering indices (it must not be passed as a
feature)
df_train = df_train.drop(['Chattering_Indices'], axis=1)
df_test = df_test.drop(['Chattering_Indices'], axis=1)

# Define the names of the columns of the two databases
COLUMNS = ["TSTR", "Y", "M", "D", "H", "m", "S", "TCOU", "TSTA", "TSNS", "SO",
           "ID", "TID", "CN", "JX", "ATSTR", "ATCOU", "ATSTA", "ATD", "VAL",
           "UNI", "CHD", "CHB"]

df_train.columns = COLUMNS
df_test.columns = COLUMNS

# Copy the Labels in different variables and drop them from the original
database
y_train = df_train.pop('CHB')  # .pop copies the Series and drop from the df
y_eval = df_test.pop('CHB')

# Define Numerical and Categorical columns
CATEGORICAL_COLUMNS = ['SO', 'ID', 'CN', 'JX', 'UNI']
NUMERIC_COLUMNS = ['Y', 'M', 'D', 'H', 'm', 'S', 'ATD', 'VAL']

# Drop unnecessary columns from the database
features = CATEGORICAL_COLUMNS + NUMERIC_COLUMNS
for element in COLUMNS[0:-2]:
    if element not in features:
        df_train.drop(columns=element, inplace=True)
        df_test.drop(columns=element, inplace=True)
```

```python
# Convert the features and store them in a list
feature_deep = []
categorical_feature = []  # categorical, must be mapped into dense
for feature_name in CATEGORICAL_COLUMNS:
    vocabulary = df_train[feature_name].unique()
    categorical_feature.append(
    tf.feature_column.categorical_column_with_vocabulary_list(feature_name,
    vocabulary))

# Convert (wrap) categorical columns into dense columns:
# Indicator
for cat_col in categorical_feature:
    feature_deep.append(tf.feature_column.indicator_column(cat_col))
# Or Embedding_column
'''
for cat_col in categorical_feature:
    feature_columns.append(tf.feature_column.embedding_column(cat_col,
dimension=9))
'''

# Add numeric features
for feature_name in NUMERIC_COLUMNS:
    feature_deep.append(tf.feature_column.numeric_column(feature_name,
    dtype=tf.float64))

# Define the Crossed features
crossed = [tf.feature_column.crossed_column(["VAL", "UNI"],
hash_bucket_size=int(1e6)),
           tf.feature_column.crossed_column(["SO", "ID", "CN"],
hash_bucket_size=int(1e6)),
           tf.feature_column.crossed_column(["SO", "ID", "JX"],
hash_bucket_size=int(1e6))]


# Input pipeline function (from DataFrame to DataSet)
def make_input_fn(data_df, label_df, num_epochs=10, shuffle=True,
batch_size=32):
    def input_function():
        ds = tf.data.Dataset.from_tensor_slices((dict(data_df), label_df))
        if shuffle:
            ds = ds.shuffle(1000)
        ds = ds.batch(batch_size).repeat(num_epochs)
        return ds

    return input_function


# Define the training and evaluation DataSets
train_input_fn = make_input_fn(df_train, y_train)
eval_input_fn = make_input_fn(df_test, y_eval, num_epochs=1, shuffle=False)

# Build the Wide&Deep Classifier (the model directory can be specified by
model_dir='<path_to_modeldir>')
classifier_wide =
tf.estimator.DNNLinearCombinedClassifier(linear_feature_columns=crossed,

dnn_feature_columns=feature_deep,

dnn_hidden_units=[1024, 512, 256])
```

```
# Train the Classifier
classifier_wide.train(train_input_fn)

# Evaluate the Classifier
results = classifier_wide.evaluate(eval_input_fn)

# Print the default metrics
clear_output()
print(results)

# Assess predictions on the evaluation database
predictions = classifier_wide.predict(eval_input_fn)  # this is a dict, the
raw prediction probabilities are stored here
expected = y_eval  # these are the real labels

# Create a DataFrame containing the predictions (threshold=0.5), the real
Labels, and the prediction probabilities
list_predict = []
list_proba = []
list_expect = []
for pred_dict, expec in zip(predictions, expected):
    class_id = pred_dict['class_ids'][0]
    probability = pred_dict['probabilities'][class_id]
    list_predict.append(class_id)
    list_proba.append(probability * 100)
    list_expect.append(expec)

df_predictions = pd.DataFrame({'Predictions': list_predict, 'Expected':
list_expect, 'Probabilities': list_proba})

# Manually calculate Tp, Fn, Fp, Recall, Precision etc.
tp = 0
fp = 0
fn = 0
tn = 0
for p, e in zip(df_predictions.Predictions, df_predictions.Expected):
    if e == 1 and p == 1:
        tp += 1
    elif p == 1 and p != e:
        fp += 1
    elif p == 0 and p != e:
        fn += 1
    elif p == 0 and e == 0:
        tn += 1

Recall = tp / (tp + fn)
Precision = tp / (tp + fp)

# Take the probabilities of a positive prediction (i.e. "1") and store it into
the DataFrame
predictions = classifier_wide.predict(eval_input_fn)
probs = pd.Series([pred['probabilities'][1] for pred in predictions])  # this
is the probability of the label being "1"
df_predictions['Prob_1'] = probs

# Save the  DataFrame as a .csv text file
df_predictions.to_csv('<path_to_save_directory>\\file_name.csv', index=False)
```

```
# Plot the ROC curve
fpr, tpr, _ = roc_curve(y_eval, probs)
plt.plot(fpr, tpr)
plt.title('ROC curve')
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.xlim(0, )
plt.ylim(0, )
plt.show()

# Obtain the confusion matrix
conf_matrix = confusion_matrix(df_predictions.Expected,
df_predictions.Predictions)
df_conf_matrix = pd.DataFrame(conf_matrix, index=[0, 1], columns=[0, 1])

# Plot the confusion matrix (parameters are optimized for A4, 3 matrices in a
row)
fig = plt.figure()
ax = fig.add_subplot(111)
fig = sns.heatmap(df_conf_matrix, annot=True, square=True, cmap='Reds',
fmt='d', annot_kws={"size": 60})
cax = plt.gcf().axes[-1]
cax.tick_params(labelsize=55)
ax.set_ylabel('True label', fontsize=70, labelpad=30)
ax.set_xlabel('Predicted label', fontsize=70, labelpad=30)
ax.tick_params(labelsize=60, axis='x')
ax.tick_params(labelsize=60, axis='y')
plt.show()

# Build precision - recall curve using a built-in function (parameters are
optimized for A4, 2 or more figures per page)
precisions_1, recalls_1, thresholds_1 =
precision_recall_curve(df_predictions.Expected, df_predictions.Prob_1)
fig_2 = plt.figure(figsize=(5, 5))
ax = fig.add_subplot(111)
ax.plot(recalls_1, precisions_1, linewidth=5)
ax.set_xlabel('Recall', fontsize=60, labelpad=30)
ax.set_xlim(0, )
ax.xaxis.set_major_locator(MultipleLocator(0.1))  # set ticks in x axis
ax.xaxis.set_minor_locator(MultipleLocator(0.05))  # set ticks in x axis
ax.set_ylim(0, )
ax.yaxis.set_major_locator(MultipleLocator(0.1))  # set ticks in x axis
ax.yaxis.set_minor_locator(MultipleLocator(0.05))  # set ticks in x axis
ax.set_ylabel('Precision', fontsize=60, labelpad=30)
ax.tick_params(which='major', direction='out', length=20, width=1, pad=20,
labelsize=40, axis='x')
ax.tick_params(which='major', direction='out', length=20, width=1, pad=20,
labelsize=40, axis='y')
ax.tick_params(which='minor', direction='out', length=10, width=1, pad=20,
axis='x')
ax.tick_params(which='minor', direction='out', length=10, width=1, pad=20,
axis='y')
ax.grid(which='major', axis='both', linestyle='-')
ax.grid(which='minor', axis='both', linestyle='--')
plt.gca().set_aspect('equal', adjustable='box')  # make the plot squared
```

```
# Build precision - recall curve manually
recalls = []
precisions = []
thresholds = list(np.arange(0, 1, 0.0001))  # a list of thresholds
for thresh in thresholds:
    list_prediction_threshold = []
    for element in df_predictions.Prob_1:
        if element >= thresh:
            list_prediction_threshold.append(1)
        else:
            list_prediction_threshold.append(0)
    conf = confusion_matrix(df_predictions.Expected,
list_prediction_threshold)
    true_p = conf[1][1]
    false_p = conf[0][1]
    false_n = conf[1][0]
    recalls.append(true_p / (true_p + false_n))
    precisions.append(true_p / (true_p + false_p))

# Store results in a DataFrame and drop nan (they rise when there are no more
Tp)
df_thresholds = pd.DataFrame({'Recall': recalls, 'Precision': precisions,
'Threshold': thresholds})
df_thresholds.dropna(inplace=True)

# Save the DataFrame
df_thresholds.to_csv('<path_to_save_directory>\\file_name.csv', index=False)
```

Appendix C

Tables

SUBSTANCE	CAS NUMBER	CLASSIFICATION (CLP Regulation)	HAZARD STATEMENTS	CONFINEMENT METHOD
Anhydrous ammonia	7664-41-7	**Flammable gas cat. 2**	**H221**	Cryogenic tank and pipes
		Gas under pressure	H280	
		Acute toxicity cat 3. (INHALATION)	**H331**	
		Skin corrosion/irritation cat. 1B	H314	
		Hazardous to the aquatic environment cat. 1	**H400**	
		Hazardous to the aquatic environment cat. 2	**H411**	
Dry natural gas (methane)	68410-63-9 (74-82-8)	**Flammable gas cat. 1**	**H220**	Pipes and plant equipment
		Gas under pressure	H280	
Sodium hypochlorite (14 – 15 %)	(7681-52-9)	Corrosive to metals	H290	Tank
		Skin corrosion/irritation cat. 1B	H314	
		Hazardous to the aquatic environment cat. 1	**H400**	
		Specific target organ toxicity (STOT) – single exposure cat. 3	H335	
Hydrogen	1333-74-0	**Flammable gas cat. 1**	**H220**	Pipes and plant equipment
		Gas under pressure	H280	
Ammonia aqueous solution (15 – 30 %)	1336-21-6	Skin corrosion/irritation cat. 1B	H314	Pipes and plant equipment
		Specific target organ toxicity (STOT) – single exposure cat. 3	H335	
		Hazardous to the aquatic environment cat. 1	**H400**	
		Hazardous to the aquatic environment cat. 2	**H411**	
Formurea 80 (aqueous solution)	50-00-0	Carcinogenicity cat. 1B	H350	Tank and pipes
		Germ cell mutagenicity cat. 2	H341	
		Acute toxicity cat 3. (**inhalation,** contact, ingestion)	H301 H311 **H331**	
		Skin corrosion/irritation cat. 2	H315	
		Skin sensitisation cat. 1	H317	
		Serious eye damage cat. 2	H319	
		Specific target organ toxicity (STOT) – single exposure cat. 3	H335	

Table C. 1– substances present in greater quantity than the thresholds defined in D.Lgs. 105/2015

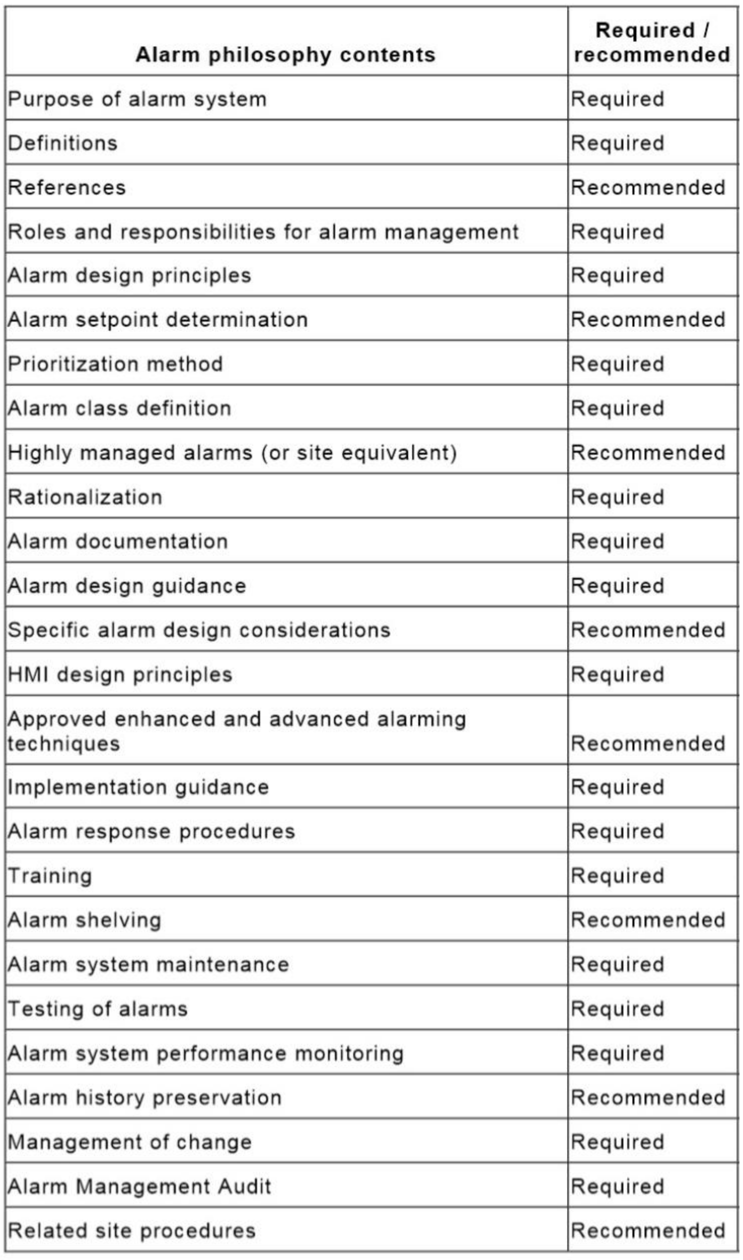
Source	Description
FI234	Mass flow rate of BFW entering E-18 (heat recovery from primary reformer exhausts). The stream leaving E-18 splits in two: part is sent to E-19 (heat recovery from the gas stream leaving the second ammonia synthesis reactor) the other part is sent to E-20A/B/D (heat recovery from the gas stream leaving high-temperature shift reactor). Part of the water leaving E-19 is sent to E-06 reboiler to produce HPS.
PI275_ESD	Pressure of the natural gas stream entering the combustion chamber of the primary reformer. If the pressure is too high, J-1 interlock sequence may be triggered, and the natural gas is sent to vent.
PI276_ESD	Pressure of the natural gas stream entering the combustion chamber of the primary reformer. If the pressure is too high or too low, J-2 interlock sequence may start the reforming section trip.
FI225A	Volumetric flow rate of natural gas entering the reaction section of the primary reformer. If the gas flow is too high, J-2 interlock sequence may start the reforming section trip.
FI225B	Same as FI225A.
FI209A	Mass flow rate of MPS entering the reaction section of the primary reformer. If the steam flow is too low, J-2 interlock sequence may start the reforming section trip.
FI209B	Same as FI209A.
TI219_4	Temperature of the gas stream leaving the primary reformer and approaching the secondary reformer. It is associated with J-1 and J-2 safety interlock logics.
FI231B	Volumetric flow rate of preheated air entering the secondary reformer. If the air flow is too low, J-3 logic may start the trip of the secondary reformer.
FI231A	Same as FI232B.
PI203	Pressure of the exhaust leaving the primary reformer combustion chamber and entering the heat recovery section. If the pressure is low (low draught), J1 and J2 logics start the trip of the reformer section and send the natural gas to vent. If the pressure is high (high draught), J-5 safety logic may stop the first reforming blower.
AI201	CH_4 concentration in the gas stream leaving the reformer section and entering the first Conversion reactor (R-04)
LI202	Water level in the steam drum D-09. It is located after the reforming section and it serves many pieces of equipment of the Conversion section: from D-09, BFW is sent to E01A/B and E03 while HPS is sent to E-02 steam superheater. If the level is low, J-1 and J-2 logics may start the trip of the reformer section and send the natural gas to vent.
LI203	Same as LI202.
FI227A	HPS volumetric flow rate leaving D-09 (discussed above) and entering E-02 steam superheater. If the flow is too low J-2 interlock sequence may start the reforming section trip.
FI227B	Same as FI227B.
AI202	CO concentration in the gas stream leaving the high-temperature shift reactor. The CO concentration must be low to avoid catalyst poisoning in the ammonia reactor. It is associated with a High concentration alarm.
LI267	BFW level in E-04 boiler. The exchanger produces LPS recovering heat from the gas stream leaving the low-temperature CO converter (R-05). The produced steam is sent to the urea section. If the level is too low/high, J-130A and J-130S safety interlock sequences may start/stop the BFW pumps.
LIC264	Same as LI267 but no associated alarms. Control function only.
LI307	Condensate level in D-01. The drum is a separator placed upstream of the Vetrocoke absorption column. From the bottom of D-01, the process condensate leaves the section, while the process gas leaves from the top of the separator and approaches the absorption column. The process condensate is used to produce demineralized water. If the level is too low, J-19 interlock logic may stop the process condensate draining.
LI308	Same as LI307.

FI306	Condensate mass flow rate from D-01 to the quench vessel D-10. The quench vessel is placed upstream of D-01 (see LI307) and, inside, the condensate coming from D-01 is sprayed to cool the process gas stream that enters the decarbonization section. The condensate is then collected in the bottom of D-10 and sent again to D-01. If the mass flow rate is too low, J-18 interlock logic may start G308A/S quenching pumps.
LI315	Level in C-01 Vetrocoke absorber. If the level is too low, J9 interlock logic stops the liquid draining, starts the absorption column and methanation reactor trip, and drains the liquid from D-02 flash vessel. If the level is too high or J9 has already been triggered, J-19 interlock logic may start the Synthesis section trip and send the converted process gas to vent.
LI328	Level in D-02 flash vessel. It is located between the absorption column and the regeneration column. It receives the liquid (CO_2 rich) from the bottom of C-01, an expansion occurs, and liquid stream partially evaporates. The gas (CO_2 rich) leaves from the top of D-02, the liquid is sent to the regeneration column C-02. If the liquid is too high, J-9 interlock logic may start the liquid draining from D-02 flash vessel, start the absorption column and methanation reactor trip, and stop the liquid draining from C-01 absorption column.
FI302	Volumetric flow of the "split" regenerated Vetrocoke solution (i.e. the one that is spilt from the bottom of C-02 and enters the top section of C-01). If the flow rate is too low, J-7 interlock logic may start the trip of the Vetrocoke pumps.
FI303	Volumetric flow of the "main" regenerated Vetrocoke solution (i.e. the one that is spilt from the centre of C-02 and enters the central section of C-01). If the flow rate is too low, J-7 interlock logic may start the trip of the Vetrocoke pumps.
FI315	Same as FI303.
FI314	Same as FI302.
PI319	Pressure of the CO_2 gas stream leaving the decarbonatization section and approaching the CO_2 compressor located in the Urea plant section. The instrument is associated with a low-pressure alarm.
PI320	Same as PI319 but J25 interlock logic is now associated. Unfortunately, the logic acts on the Urea plant; thus, the documentation is not available.
LI335A	Level in D-11 separator vessel, it is the absorption column top separator (the "clean" process gas stream that leaves from the top of the absorption column contains liquid, that must be removed). If the level is low, J-43 interlock logic may stop the liquid draining.
LI335B	Same as LI335B.
LIC318	Level in D-12 separator. It is placed after the methanation section and it collects and removes the last portion of condensate from the process stream. After D-12 the syngas is sent to the compressor P-01. The instrument has control purpose and it is associated with a high-level alarm.
LI319	Same as LIC318 but it is associated with interlock logics. If the level is too high, J6 interlock logic starts the trip of Metanator. If the level is too low, J-27 may stop the draining of the process condensate.
PI3400	Pressure of the syngas entering the first stage of P-01 compressor. A low-pressure alarm is present, no safety interlock logics are associated.
PI3410	Same as PI3400 but, if the pressure is too low, J-434 interlock logic may start the trip of the syngas compressor.
PI3415	Pressure of the lubricant serving P-01 compressor. If the pressure is too low, J-432 may start the pump G432s.
PI3415B	Same as PI3415.
PI3427	Pressure of the vapour extracted from FTP-01 steam turbine (it drives the syngas compressor). If the pressure is too high, J434 interlock logic starts the trip of the compressor. If the pressure is too low, J-422 may start the trip of the P-03 compressor.
PI3429A	Pressure of the MPS extracted from FTP-01 turbine and sent to FTP-03 (it drives the air compressor P-03). If the pressure is too low, J-422 interlock logic may start the trip of the P-03 compressor.

PI3429B	Same as PI3429B.
PDI3408A	Pressure difference between the P-01 fifth stage outlet (makeup syngas) and the P-01 sixth stage outlet (recycled syngas). Basically, it measures the pressure difference between the two syngas streams that are mixed and sent to the ammonia synthesis section. If the pressure difference is too high, J-434 interlock logic may start the trip of the syngas compressor.
PDI3408B	Same as PDI3408A.
PI501	Pressure of the syngas stream entering the first ammonia synthesis reactor. If the pressure is too high, J-28 interlock logic may start the trip of B-02.
LI585	BFW level in E-06. It produces HPS recovering heat from the process gas leaving the second ammonia reactor. If the level is too low or too high, J-435 interlock logic may unload P-01 syngas compressor.
LI585B	Same as LI585.
LI585C	Same as LI585.
LI503	Level in the D-03 ammonia separator. It is located after the ammonia cooling section where the gas stream leaving the second ammonia reactor is cooled and partially condensed. From the top of D-03 the gas (syngas) is recycled back to the compressor. From the bottom of D-03 the liquid (mainly ammonia) is sent to the expansion section. If the level is too high, J-435 interlock logic may unload P-01 syngas compressor.
LI414	Level in the wash column C-03. In the column, ammonia is used to remove the last traces of oxygenated compound from the syngas. The washing column is placed between the fourth and the fifth of the syngas compressor. If the level is too high, J-434 interlock logic may start the trip of the syngas compressor.
LI415	Same as LI414.
PI4422	Pressure of the lubricant serving FTP-02 steam turbine (it drives the ammonia compressor P-02). If the pressure is too low, J-439 may stop the ammonia compressor.
PI4431	Same as PI4422.
PI4432	Same as PI4422.
PI1570A	Pressure of the anhydrous ammonia that flows from D-05 flash vessel (the last vessel of the expansion section) to the ammonia storage vessel (D-06). If the pressure is low, J-758 interlock logic may stop the ammonia pumping.
PI1570B	Same as PI1570A.
PI1570C	Same as PI1570A.

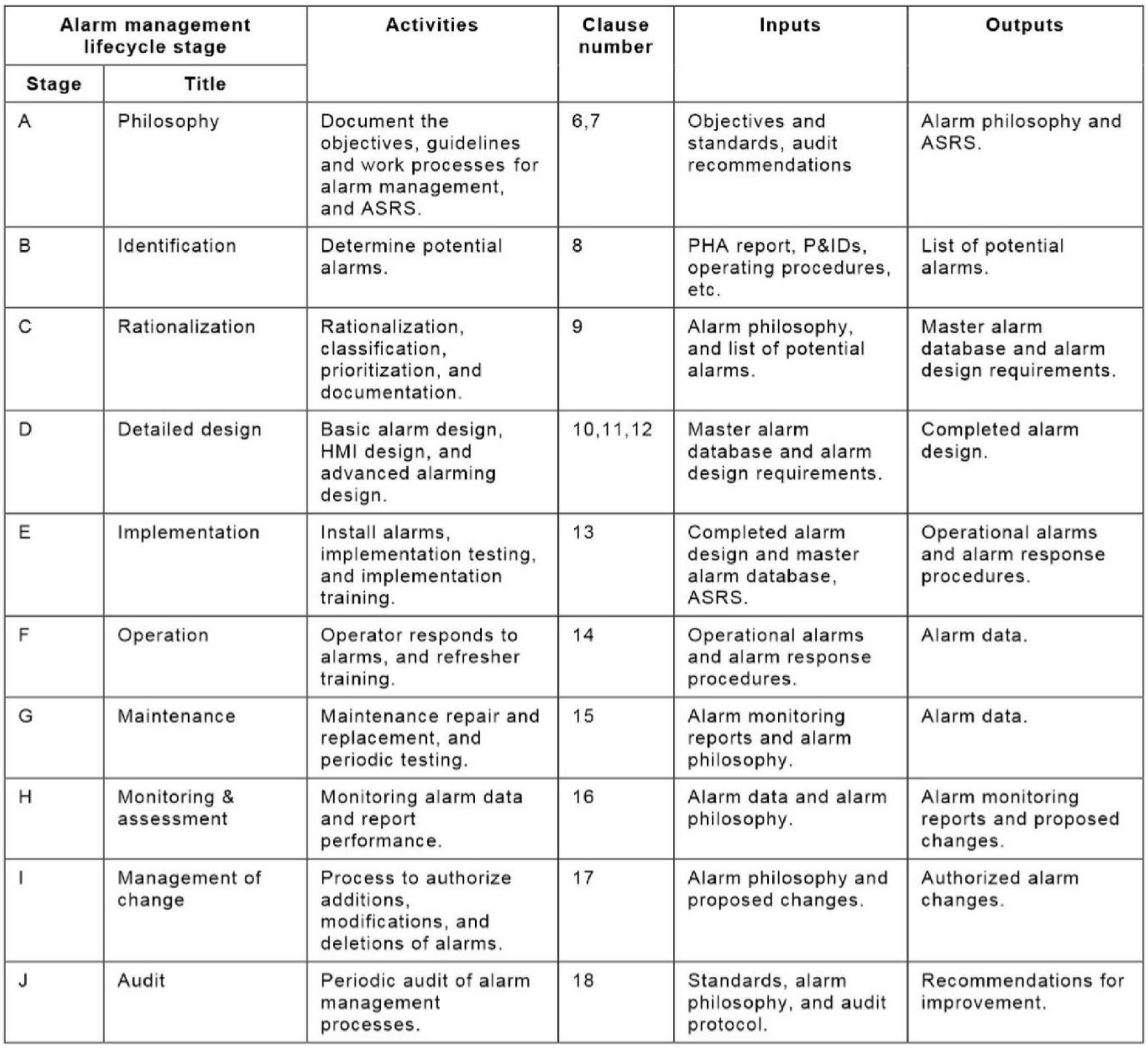
Table C. 2 - Alarm location

Remark: the expression "J-xxx ***may*** stop/start/unload etc." implies that, usually, the interlock logics require more than one condition to be activated. Thus, if just one condition is met, it may not be enough to activate the logic.



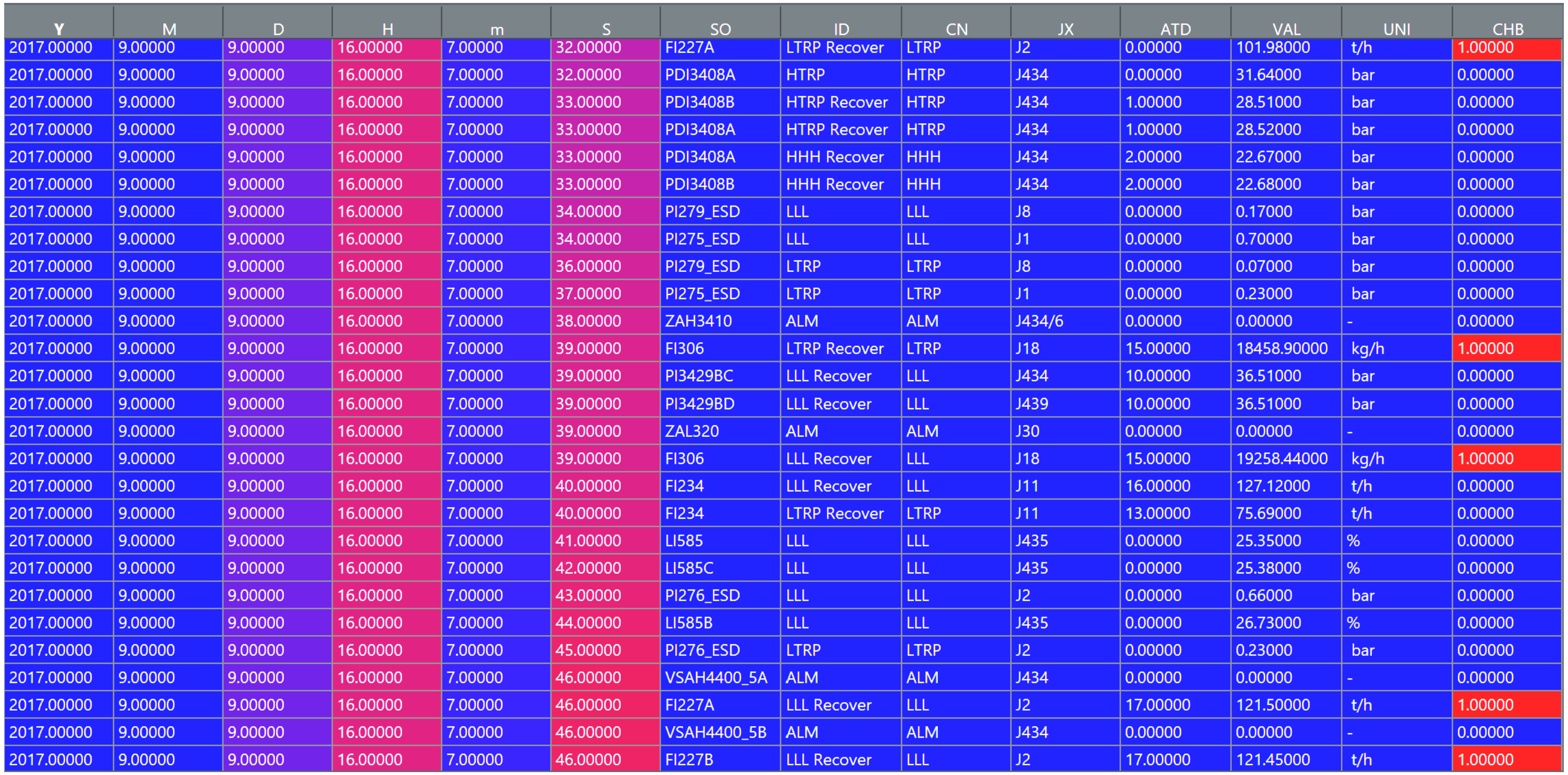
Alarm philosophy contents	Required / recommended
Purpose of alarm system	Required
Definitions	Required
References	Recommended
Roles and responsibilities for alarm management	Required
Alarm design principles	Required
Alarm setpoint determination	Recommended
Prioritization method	Required
Alarm class definition	Required
Highly managed alarms (or site equivalent)	Recommended
Rationalization	Required
Alarm documentation	Required
Alarm design guidance	Required
Specific alarm design considerations	Recommended
HMI design principles	Required
Approved enhanced and advanced alarming techniques	Recommended
Implementation guidance	Required
Alarm response procedures	Required
Training	Required
Alarm shelving	Recommended
Alarm system maintenance	Required
Testing of alarms	Required
Alarm system performance monitoring	Required
Alarm history preservation	Recommended
Management of change	Required
Alarm Management Audit	Required
Related site procedures	Recommended

Table C. 3 - Required and recommended alarm philosophy content (ANSI ISA, 2016)



Alarm management lifecycle stage		Activities	Clause number	Inputs	Outputs
Stage	Title				
A	Philosophy	Document the objectives, guidelines and work processes for alarm management, and ASRS.	6,7	Objectives and standards, audit recommendations	Alarm philosophy and ASRS.
B	Identification	Determine potential alarms.	8	PHA report, P&IDs, operating procedures, etc.	List of potential alarms.
C	Rationalization	Rationalization, classification, prioritization, and documentation.	9	Alarm philosophy, and list of potential alarms.	Master alarm database and alarm design requirements.
D	Detailed design	Basic alarm design, HMI design, and advanced alarming design.	10,11,12	Master alarm database and alarm design requirements.	Completed alarm design.
E	Implementation	Install alarms, implementation testing, and implementation training.	13	Completed alarm design and master alarm database, ASRS.	Operational alarms and alarm response procedures.
F	Operation	Operator responds to alarms, and refresher training.	14	Operational alarms and alarm response procedures.	Alarm data.
G	Maintenance	Maintenance repair and replacement, and periodic testing.	15	Alarm monitoring reports and alarm philosophy.	Alarm data.
H	Monitoring & assessment	Monitoring alarm data and report performance.	16	Alarm data and alarm philosophy.	Alarm monitoring reports and proposed changes.
I	Management of change	Process to authorize additions, modifications, and deletions of alarms.	17	Alarm philosophy and proposed changes.	Authorized alarm changes.
J	Audit	Periodic audit of alarm management processes.	18	Standards, alarm philosophy, and audit protocol.	Recommendations for improvement.

Table C. 4 - Alarm management lifecycle stage inputs and outputs (ANSI ISA, 2016)



Y	M	D	H	m	S	SO	ID	CN	JX	ATD	VAL	UNI	CHB
2017.00000	9.00000	9.00000	16.00000	7.00000	32.00000	FI227A	LTRP Recover	LTRP	J2	0.00000	101.98000	t/h	1.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	32.00000	PDI3408A	HTRP	HTRP	J434	0.00000	31.64000	bar	0.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	33.00000	PDI3408B	HTRP Recover	HTRP	J434	1.00000	28.51000	bar	0.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	33.00000	PDI3408A	HTRP Recover	HTRP	J434	1.00000	28.52000	bar	0.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	33.00000	PDI3408A	HHH Recover	HHH	J434	2.00000	22.67000	bar	0.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	33.00000	PDI3408B	HHH Recover	HHH	J434	2.00000	22.68000	bar	0.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	34.00000	PI279_ESD	LLL	LLL	J8	0.00000	0.17000	bar	0.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	34.00000	PI275_ESD	LLL	LLL	J1	0.00000	0.70000	bar	0.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	36.00000	PI279_ESD	LTRP	LTRP	J8	0.00000	0.07000	bar	0.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	37.00000	PI275_ESD	LTRP	LTRP	J1	0.00000	0.23000	bar	0.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	38.00000	ZAH3410	ALM	ALM	J434/6	0.00000	0.00000	-	0.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	39.00000	FI306	LTRP Recover	LTRP	J18	15.00000	18458.90000	kg/h	1.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	39.00000	PI3429BC	LLL Recover	LLL	J434	10.00000	36.51000	bar	0.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	39.00000	PI3429BD	LLL Recover	LLL	J439	10.00000	36.51000	bar	0.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	39.00000	ZAL320	ALM	ALM	J30	0.00000	0.00000	-	0.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	39.00000	FI306	LLL Recover	LLL	J18	15.00000	19258.44000	kg/h	1.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	40.00000	FI234	LLL Recover	LLL	J11	16.00000	127.12000	t/h	0.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	40.00000	FI234	LTRP Recover	LTRP	J11	13.00000	75.69000	t/h	0.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	41.00000	LI585	LLL	LLL	J435	0.00000	25.35000	%	0.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	42.00000	LI585C	LLL	LLL	J435	0.00000	25.38000	%	0.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	43.00000	PI276_ESD	LLL	LLL	J2	0.00000	0.66000	bar	0.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	44.00000	LI585B	LLL	LLL	J435	0.00000	26.73000	%	0.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	45.00000	PI276_ESD	LTRP	LTRP	J2	0.00000	0.23000	bar	0.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	46.00000	VSAH4400_5A	ALM	ALM	J434	0.00000	0.00000	-	0.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	46.00000	FI227A	LLL Recover	LLL	J2	17.00000	121.50000	t/h	1.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	46.00000	VSAH4400_5B	ALM	ALM	J434	0.00000	0.00000	-	0.00000
2017.00000	9.00000	9.00000	16.00000	7.00000	46.00000	FI227B	LLL Recover	LLL	J2	17.00000	121.45000	t/h	1.00000

Table C. 5 - Part of the database used for the TensorFlow simulations

Bibliography

Abadi, M. *et al.* (2016) 'TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems'. Available at: http://arxiv.org/abs/1603.04467.

Ahmed, K. *et al.* (2013) 'Similarity analysis of industrial alarm flood data', *IEEE Transactions on Automation Science and Engineering*. IEEE, 10(2), pp. 452–457. doi: 10.1109/TASE.2012.2230627.

Aika, K. *et al.* (2012) *Ammonia: catalysis and manufacture*. Springer Science & Business Media.

AL-Dhfeery, A. A. and Jassem, A. A. (2012) 'Modeling and simulation of an industrial secondary reformer reactor in the fertilizer plants', *International Journal of Industrial Chemistry*, 3(1), pp. 1–8. doi: 10.1186/2228-5547-3-14.

ANSI ISA (2016) 'ANSI/ISA–18.2–2016 Management of Alarm Systems for the Process Industries', *ANSI ISA*.

Arpae (2019) *Rischio industriale | Arpae*. Available at: https://www.arpae.it/elenchi_dinamici.asp?tipo=rir_fe&idlivello=111 (Accessed: 24 January 2020).

Brink, H., Richards, J. and Fetherolf, M. (2016) *Real-World Machine Learning*. Manning Publications. Available at: https://books.google.no/books?id=DoQAswEACAAJ.

Cheng, H.-T. *et al.* (2016) 'Wide & deep learning for recommender systems', in *Proceedings of the 1st workshop on deep learning for recommender systems*, pp. 7–10.

D.Lgs. 105/2015 (no date) *Attuazione della direttiva 2012/18/UE relativa al controllo del pericolo di incidenti rilevanti con- nessi con sostanze pericolose.*

EEMUA (2013) 'EEMUA Publication 191 Alarm systems - a guide to design, management and procurement'.

GitHub.com (2020) *GitHub - tensorflow/tensorflow: An Open Source Machine Learning Framework for Everyone*. Available at: https://github.com/tensorflow/tensorflow (Accessed: 24 January 2020).

Google (2020a) *Classification: Accuracy | Machine Learning Crash Course*. Available at: https://developers.google.com/machine-learning/crash-course/classification/accuracy (Accessed: 24 January 2020).

Google (2020b) *Classification: Precision and Recall | Machine Learning Crash Course*. Available at: https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall (Accessed: 24 January 2020).

Google (2020c) *Feature Crosses | Machine Learning Crash Course | Google Developers*. Available at: https://developers.google.com/machine-learning/crash-course/feature-crosses/video-lecture (Accessed: 24 January 2020).

Han, J., Kamber, M. and Pei, J. (2012) *Data Mining: Concepts and Techniques*, *Data Mining: Concepts and Techniques*. Elsevier Inc. doi: 10.1016/C2009-0-61819-5.

Hastie, T., Friedman, R. and Tibshirani, J. (2009) *The Elements of Statistical Learning*. Springer-Verlag New York. doi: 10.1007/978-0-387-84858-7.

Health and Safety Executive (1997) *The Explosion and Fires at the Texaco Refinery, Milford Haven, 24 July 1994: A Report of the Investigation by the Health and Safety Executive Into the Explosion and Fires on the Pembroke Cracking Company Plant at the Texaco Refinery, Milford Haven on 24 J*. HSE Books (Incident Report Series). Available at: https://books.google.no/books?id=Y4QxNQAACAAJ.

Hu, W. *et al.* (2015) 'An application of advanced alarm management tools to an oil sand extraction plant', *IFAC-PapersOnLine*, 28(8), pp. 641–646. doi: 10.1016/j.ifacol.2015.09.040.

James, G. *et al.* (2013) *An Introduction to Statistical Learning: With Applications in R*. Springer-Verlag New York. doi: 10.1007/978-1-4614-7138-7.

Jennings, J. R. (1991) *Catalytic Ammonia Synthesis*, *Springer Science & Business Media*. doi: 10.1007/978-1-4757-9592-9.

Katzel, J. (2007) *Control Engineering | Managing Alarms*. Available at: https://www.controleng.com/articles/managing-alarms/ (Accessed: 23 January 2020).

Kondaveeti, S. R. *et al.* (2010) *Graphical representation of industrial alarm data*, *IFAC Proceedings Volumes (IFAC-PapersOnline)*. IFAC. doi: 10.3182/20100831-4-fr-2021.00033.

Kondaveeti, S. R. *et al.* (2013) 'Quantification of alarm chatter based on run length distributions', *Chemical Engineering Research and Design*. Institution of Chemical Engineers, 91(12), pp. 2550–2558. doi: 10.1016/j.cherd.2013.02.028.

Laberge, J. C. *et al.* (2014) 'Addressing alarm flood situations in the process industries through alarm summary display design and alarm response strategy', *International Journal of Industrial Ergonomics*. Elsevier Ltd, 44(3), pp. 395–406. doi: 10.1016/j.ergon.2013.11.008.

Lesot, M. J., Rifqi, M. and Benhadda, H. (2009) 'Similarity measures for binary and numerical data: a survey', *International Journal of Knowledge Engineering and Soft Data Paradigms*, 1(1), p. 63. doi: 10.1504/ijkesdp.2009.021985.

Liu, J. *et al.* (2018) 'Artificial intelligence in the 21st century', *IEEE Access*, 6(April), pp. 34403–34421. doi: 10.1109/ACCESS.2018.2819688.

livelibrary.osisoft.com (2020) *PI Server 2014 - Deadband*. Available at: https://livelibrary.osisoft.com/LiveLibrary/content/en/server-v3/GUID-886A8250-B00A-4528-B6DE-8C86F086E722#addHistory=true&filename=GUID-1085834D-E463-4E72-9A00-23C0EB56D2D4.xml&docid=GUID-886A8250-B00A-4528-B6DE-8C86F086E722&inner_id=&tid=&query=&scope=&resource=&toc=false&eventType=lcContent.loadDocGUID-886A8250-B00A-4528-B6DE-8C86F086E722 (Accessed: 27 January 2020).

Mohri, M., Rostamizadeh, A. and Talwalkar, A. (2012) *Foundations of Machine Learning*. MIT Press (Adaptive Computation and Machine Learning series). Available at: https://books.google.no/books?id=maz6AQAAQBAJ.

Pattabathula, V. and Richardson, J. (2016) 'Introduction to ammonia production', *Chemical Engineering Progress*, 112(9), pp. 69–75.

Ravi, R. and Wu, L.-C. (2016) *DEMYSTIFYING INDUSTRY 4.0: IMPLICATIONS OF INTERNET OF THINGS AND SERVICES FOR THE CHEMICAL INDUSTRY*, *MIT Global Scale Network*. Available at: https://dspace.mit.edu/bitstream/handle/1721.1/102155/2015_11_Ravi_Wu.pdf?sequence=1.

Reis, M. S. and Kenett, R. (2018) 'Assessing the value of information of data-centric activities in the chemical processing industry 4.0', *AIChE Journal*, 64(11), pp. 3868–3881. doi: 10.1002/aic.16203.

Samuel, A. L. (1959) 'Some studies in machine learning using the game of checkers', *IBM Journal of Research and Development*, 44(1–2), pp. 207–219. doi: 10.1147/rd.441.0206.

Santini, D. (2018) *A Prevention Approach for Chemical Process Risk Management: Retrieving Knowledge from Relevant Databases Through Machine Learning*. UNIBO, NTNU.

Shaw, J. A. (1993) 'DCS-based alarms: Integrating traditional functions into modern technology', *ISA Transactions*, 32(2), pp. 177–181. doi: 10.1016/0019-0578(93)90039-Y.

Stanton, N. A. and Barber, C. (1995) ‘Alarm-initiated activities: an analysis of alarm handlingby operators using text-based alarm systems in supervisory control systems’, *Ergonomics*. Taylor & Francis, 38(11), pp. 2414–2431. doi: 10.1080/00140139508925276.

TensorFlow.org (2020a) *API Documentation | TensorFlow Core r2.1*. Available at: https://www.tensorflow.org/api_docs (Accessed: 24 January 2020).

TensorFlow.org (2020b) *Case studies | TensorFlow*. Available at: https://www.tensorflow.org/about/case-studies/ (Accessed: 24 January 2020).

TensorFlow.org (2020c) *Classify structured data with feature columns | TensorFlow Core*. Available at:https://www.tensorflow.org/tutorials/ structured_data/feature_columns (Accessed: 24 January 2020).

TensorFlow.org (2020d) *Install | TensorFlow*. Available at: https://www.tensorflow.org/install?hl=it (Accessed: 24 January 2020).

TensorFlow.org (2020e) *Models and layers | TensorFlow.js*. Available at: https://www.tensorflow.org/js/guide/models_and_layers (Accessed: 24 January 2020).

Topsoe.com (2020) *SynCOR™ technology elements | Haldor Topsoe*. Available at: https://www.topsoe.com/syncortm-technology-elements (Accessed: 24 January 2020).

Yang, F. *et al.* (2012) ‘Improved correlation analysis and visualization of industrial alarm data’, *ISA Transactions*. Elsevier Ltd, 51(4), pp. 499–506. doi: 10.1016/j.isatra.2012.03.005.

Yara.it (2020) *Ferrara | Yara Italia*. Available at: https://www.yara.it/chi-siamo/yara-italia/ferrara/ (Accessed: 24 January 2020).

Yara Italia S.p.A (2016) *Relazione di riferimento della Yara Italia S.p.A. dello stabilimento di Ferrara*. Available at: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=4&ved=2ahUKEwjo37_N3pfnAhUvlosKHd9OC1MQFjADegQIBRAB&url=https%3A%2F%2Fva.minambiente.it%2FFile%2FDocumento%2F283547&usg=AOvVaw0SH3KqAZHIyF6IJBXY0vjZ.

Nicola Tamascelli

NTNU
Norwegian University of
Science and Technology