COPPE
Instituto Alberto Luiz Coimbra de
Pós-Graduação e Pesquisa de Engenharia
UFRJ

SIMILARITY-BASED METHODS FOR MACHINE DIAGNOSIS

Felipe Moreira Lopes Ribeiro

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

Orientadores: Eduardo Antônio Barros da Silva
Sergio Lima Netto

Rio de Janeiro
Agosto de 2018

SIMILARITY-BASED METHODS FOR MACHINE DIAGNOSIS

Felipe Moreira Lopes Ribeiro

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

Examinada por:

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

Prof. Sergio Lima Netto, Ph.D.

Prof. Diego Barreto Haddad, D.Sc.

Prof. João Baptista de Oliveira e Souza Filho, D.Sc.

Dr. Mario Cesar Mello Massa de Campos, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
AGOSTO DE 2018

Ribeiro, Felipe Moreira Lopes

Similarity-based methods for machine diagnosis/Felipe Moreira Lopes Ribeiro. – Rio de Janeiro: UFRJ/COPPE, 2018.

XVII, 132 p.: il.; 29, 7cm.

Orientadores: Eduardo Antônio Barros da Silva
Sergio Lima Netto

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2018.

Referências Bibliográficas: p. 105 – 112.

1. Diagnosis. 2. Machinery Fault Detection. 3. Machine Learning. I. Silva, Eduardo Antônio Barros da *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

To my family and friends

Acknowledgments

This is my first time writing my acknowledgments in English. English is not my first language by birth, but my first language by choice. And while I really would like to write something grandiloquent to appease future readers, there is little inspiration in this text, but a mix of tension, like a stage fright, some satisfaction, just a little, and a lot of gratitude.

Tension and worries when I look at the future, and during this writing. There are many examples of theses and dissertations everywhere where someone can find his ‘inspiration’. There is a lot of doubt about the best way to proceed, what to write and how to start. But at the end, the best choice is to take your time, delve into the text, and hope for the best.

There is satisfaction for another step made. Baby step, surely, but a step forward, nevertheless. Not all that was in my mind made to this text? True. “Time is the master and time can be a disaster!”. Many problems, mistakes, and less successful approaches were left behind. But maybe tomorrow, or some day in the future, maybe they will have their time to shine. But for now, there is this small stream of satisfaction of completing another stage to a (hopefully) better future.

Lastly, gratitude. Without a little (or a lot of) help of my friends and family, this text would not be there, neither the ideas that made this work possible. So first, I would like to thank my family, my parents Tania and Sebastião, my first and foremost examples, who gave me the love and necessary support to be here. My relatives, who shared good and bad moments, side by side. My sweetheart Bruna. And my extended family, who I hope to share with her someday.

Then, my friends. First, the ones from early. Bruno, Euler e Rodrigo. May the Force be with you. My friends from undergrad: Claudio, Danilo, Jaque, Simão, Vitor, Zheng. Live long and prosper. My friends from the SMT. Starting from the first generation: Amaro, Anderson, Andreas, Gabriel, Markus, Prego, Tadeu, Wallace, and Zé. The second: Allan, Aninha, Camila, Lucas, Luiz, Luis Lucas, Jonathan, and Renam. And the third: Barbosa, Cinelli, Igor, Matheus, Rafael, Roberto, and Wesley. My friends, you bow to no one. For the next generation to come, good night and good luck.

Also, to my colleagues from OLX who, like J. Fernandes Pinto from the poem

"Quadrilha", were not in the story, but became part of my life in the last year. This was an eventful year, with a lot of learning and some fun. Thanks for the opportunity of working with great people.

Lastly, but no least, my advisors, Eduardo and Sergio, thanks for the helping hand, ideas, advices, and patience. Thank for believing in my potential. And for the examiners, thanks for your help and the paid attention. Hopefully, this work should contribute with some ideas. And for the future readers, thanks for reading even the acknowledgments.

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

MÉTODOS BASEADOS EM SIMILARIDADE PARA DIAGNÓSTICO DE MÁQUINAS

Felipe Moreira Lopes Ribeiro

Agosto/2018

Orientadores: Eduardo Antônio Barros da Silva
Sergio Lima Netto

Programa: Engenharia Elétrica

Este trabalho apresenta um sistema de manutenção preditiva para diagnóstico automático de falhas em máquinas. O sistema proposto, baseado em uma técnica denominada *similarity-based modeling* (SBM), provê informações sobre o estado atual do equipamento (grau de anomalia), e retorna um conjunto de amostras representativas que pode ser utilizado para descrever o estado atual de forma esparsa, permitindo a um operador avaliar a melhor decisão a ser tomada. O sistema é modular e agnóstico aos dados, permitindo que seja utilizado em variados equipamentos e dados com pequenas modificações. As principais contribuições deste trabalho são: o estudo abrangente da proposta do classificador SBM multi-classe e o seu uso em diferentes bases de dados, seja como um classificador ou auxiliando outros classificadores comumente usados; novos métodos para a seleção de amostras representativas para os modelos SBM; o uso de novas funções de similaridade; e um serviço de detecção de falhas pronto para ser utilizado em produção. Essas contribuições atingiram o objetivo de melhorar o desempenho dos modelos SBM em cenários de classificação de falhas e reduziram sua complexidade computacional. O sistema proposto foi avaliado em três bases de dados, atingindo desempenho igual ou superior ao desempenho de trabalhos anteriores nas mesmas bases. Comparações com outros métodos são apresentadas para a recém desenvolvida *Machinery Fault Database* (MaFaulDa) e para a base de dados da *Case Western Reserve University* (CWRU). As técnicas propostas melhoraram a capacidade de generalização dos modelos de similaridade e do classificador final, atingindo acurácias de 98.5% na MaFaulDa e 98.9% na base de dados CWRU. Esses resultados apontam que a abordagem proposta baseada na técnica SBM tem potencial para ser investigada em mais profundidade.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

SIMILARITY-BASED METHODS FOR MACHINE DIAGNOSIS

Felipe Moreira Lopes Ribeiro

August/2018

Advisors: Eduardo Antônio Barros da Silva
Sergio Lima Netto

Department: Electrical Engineering

This work presents a data-driven condition-based maintenance system based on similarity-based modeling (SBM) for automatic machinery fault diagnosis. The proposed system provides information about the equipment current state (degree of anomaly), and returns a set of exemplars that can be employed to describe the current state in a sparse fashion, which can be examined by the operator to assess a decision to be made. The system is modular and data-agnostic, enabling its use in different equipment and data sources with small modifications. The main contributions of this work are: the extensive study of the proposition and use of multiclass SBM on different databases, either as a stand-alone classification method or in combination with an off-the-shelf classifier; novel methods for selecting prototypes for the SBM models; the use of new similarity functions; and a new production-ready fault detection service. These contributions achieved the goal of increasing the SBM models performance in a fault classification scenario while reducing its computational complexity. The proposed system was evaluated in three different databases, achieving higher or similar performance when compared with previous works on the same database. Comparisons with other methods are shown for the recently developed Machinery Fault Database (MaFaulDa) and for the Case Western Reserve University (CWRU) bearing database. The proposed techniques increase the generalization power of the similarity model and of the associated classifier, having accuracies of 98.5% on MaFaulDa and 98.9% on CWRU database. These results indicate that the proposed approach based on SBM is worth further investigation.

Contents

List of Figures

List of Tables

List of Algorithms

Glossary

AAKR auto-associative kernel regression.

ABVT alignment-balance-vibration trainer.

ACC accuracy.

ANN artificial neural network.

CBM condition-based maintenance.

CWRU Case Western Reserve University.

DFT discrete Fourier transform.

EVR explained variance regression.

HTTP Hypertext Transfer Protocol.

IQR inter-quartile range.

JSON JavaScript Object Notation.

$k$nn k-nearest neighbors.

MAE mean absolute error.

MaFaulDa machinery fault database.

MFS machinery fault simulator.

MSE mean Squared error.

PPV positive predictive value.

R^2 coefficient of determination.

REST Representational State Transfer.

RF random forest.

RUL remaining useful life.

SBM similarity-based modeling.

SI Système international.

SVM support vector machine.

TPR true positive rate.

Chapter 1

Introduction

Nowadays, machine learning methods and techniques permeate our daily lives. Detecting far away stars in astronomy, learning our shopping behaviors, identifying suspects, finding the best route to home, or even finding a date, machine learning is everywhere. Similarly to what happened during the industrial revolution, as data-driven systems become ubiquitous, they are changing the ways of how we interact with home appliances, work tools, or even with one another, with positive and negative impacts.

The microelectronics revolution that provided the means for the machine learning popularization also made multiple sensors, storage, computational power, and communication means accessible for cheaper applications. This made available a considerable volume of data and the computational means for learning algorithms for different purposes.

Equipment faults and failures are other common occurrences in our daily lives. As time passes by, an equipment can suffer damage from usage, misusage, rust, impact, severe weather, among other causes, which can compromise its normal operation. To recover the equipment or even reduce these occurrences, maintenance is necessary. However, while maintenance is necessary, it is not always a desirable process, as during maintenance an equipment would be unavailable and it can require spare parts or a specialist support. These options increase costs and reduce the user overall satisfaction and confidence in the equipment. As equipment are gradually becoming more complex and requiring high availability, monitoring their health with multiple sensors and data analytics can reduce unneeded maintenance stops and overall maintenance costs, increasing user satisfaction.

Given this context, the objective of this work is to produce a data-driven method based on machine learning to detect, identify, and, hopefully, predict possible faults in any equipment. This method should be flexible and operate with multiple equipment and data of distinct sources. This *predictive maintenance*, also known as condition-based maintenance [1, 2], has the task of helping any operator in planing and executing maintenance only when it is necessary, reducing maintenance cost and increasing equipment life time.

Such a system should provide information about the current degree of the anomaly, and should be easily *interpretable* by an operator. To satisfy these requirements, in this work we employed a methodology presented in [3] named similarity-based modeling. Roughly, in this methodology a sample is evaluated by its similarity with sets of selected exemplars. The new sample state[1] is assumed to be shared by the most similar set, and the degree of similarity denotes the confidence in this assumption. Also, the sets of exemplars can also be used to describe the state in a sparse fashion and can be examined by the operator to assess if corrected decision was taken [4], thus satisfying the proposed requirements.

1.1 Outline and contributions

In this document we propose a data-driven condition-based maintenance system based on similarity for diagnosis of machinery faults. This system should be agnostic, in a sense that the data sources or the current equipment being evaluated should be transparent for the system. This requirement implies some flexibility into the system, which we tried to accomplish by following a modular approach similar to the ones presented in [1] and [2]. The present document has the following contributions:

- Extensions to the original similarity-based modeling methodology, including new similarity metrics/functions, novel methods for selecting the exemplar samples, and a multiclass extension for the original binary methodology;
- A qualitative and quantitative study of the methodology when applied to the case of rotating machine databases [5, 6];
- A study of the deployment of the current system over an industrial application, which was done step-by-step, from the data acquisition to the analysis of results;
- Creation of a web application for detecting, analyzing, and monitoring possible anomalous events in a given equipment, supporting operational decisions.

The presented discussion is organized as follows. Chapter 2 introduces the data-driven methodology for condition-based maintenance. This chapter introduces the methodology used in this work, including the proposed system components, concepts

[1] In this work a *state* is a functional condition of a system that can be discriminated from other conditions given its measured *sample state*, the set of known sensors and measurements. The concepts of sample state and state are sometimes used as synonymous during this work.

of machine learning, metrics used to evaluated the generated models, and a brief discussion of the implemented framework and the datasets used during this work.

Chapter 3 introduces the similarity models used for detecting and identifying faults and failures in the monitored equipment. It also presents the main contributions of this work: the new similarity functions; the new methods for selecting the representative set of samples; and the multiclass extension for the original classifier.

A study case using the proposed methodology on rotating-machinery databases is presented in Chapter 4. This study includes multiple experiments evaluating alternative versions of the similarity-based modeling approach, the effect of different features on the system, the use of the model as a feature generator for another classifier, and the influence of the different parameters on the system performance. It also compares the proposed framework with previous works on the same databases.

Another empirical study in a real application over data from an oil and gas industry equipment is presented in Chapter 5. This chapter presents all steps of a data-driven problem: data acquisition, data preprocessing, exploratory data analysis, model training, and, lastly, the obtained conclusions.

Chapter 6 presents the proposed techniques implemented as part of a web service for anomalous event detection and monitoring, and the implemented framework, including its components, user cases, and functionalities.

Chapter 7 closes this document presenting the conclusions and the future directions of this work.

1.2 Publications

The following publications are directly related with this work

- MARINS, M. A., RIBEIRO, F. M. L., NETTO, S. L., da SILVA, E. A. B., "Improved similarity-based modeling for the classification of rotating-machine failures," *Journal of the Franklin Institute*, v. 355, n. 4, pp. 1913–1930, July 2018.

- RIBEIRO, F. M. L., MARINS, M. A., NETTO, S. L., da SILVA, E. A. B., "Rotating machinery fault diagnosis using similarity-based models," In: *Proc. Simpósio Brasileiro de Telecomunicações e Processamento de Sinais*, September 2017.

And following were concluded during the D.Sc. interval

- RIBEIRO, F. M. L., de OLIVEIRA, J. F. L., CIANCIO, A. G., da SILVA, E. A. B., ESTRADA, C. R. D., TAVARES, L. G. C., GOIS, J. N., SAID, A., MARTELOTTE, M. C., "Quality of Experience in a Stereoscopic Multiview

Environment," in *IEEE Transactions on Multimedia*, vol. 20, no. 1, pp. 1–14, January 2018.

- ARAUJO, G. M., RIBEIRO, F. M. L., JÚNIOR, W. S. S., da SILVA, E. A. B., GOLDENSTEIN, S. K. , "Weak Classifier for Density Estimation in Eye Localization and Tracking," in *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3410–3424, July 2017.

Chapter 2

Condition-based maintenance

2.1 Introduction

Failure is the lack of ability of a system to perform its intended function as designed. Failure may be the result of one or many faults. A *fault* is an abnormal condition or defect at a system which may lead to failure [7].

The process of performing actions to keep a system in working order, avoiding possible faults or restoring it to a functional state is named *maintenance*. Maintenance is an expensive process: a maintenance team must be mobilized; equipment can become unavailable during maintenance; spare parts can be needed or consumed; or it can arouse new faults. As such, maintenances should ideally occur to keep equipment and process at their best conditions, to suit security and operational specifications, and to reduce costs [2].

One of the principal causes of concern in the energy and oil industry is the maintenance of critical equipment to ensure high levels of reliability, availability, and performance [8]. To meet this aim, numerous maintenance strategies were devised, which can be categorized in three main groups [1, 2]:

1. Corrective maintenance;
2. Preventive maintenance; and
3. Predictive maintenance.

Corrective maintenance is a reactive type of maintenance in which the system is assumed to be in its appropriate condition until proven otherwise. Maintenance is performed only when a failure occurs. This strategy eliminates unnecessary maintenances cost at the price of high risk of collateral damage, high production downtime, and high cost of spare parts, as it ensures that the system will fail [1, 2].

Conversely, at the preventive or *periodic* maintenance an optimum breakdown window is computed, based on the system operational conditions and maintenance and operational costs, and routine maintenances activities are scheduled to prevent failure from occurring, assuming that failure will occur otherwise [1, 2]. While these activities may minimize operating costs, as they reduce the number of unnecessary stops and the number of failures, and produce greater control over the performed maintenance, they typically involve the highest maintenance costs and there will still be unpredicted failures. Currently, this is the most popular strategy, actively being employed by manufacturers and operators in industry [2].

Predictive maintenance, also known as condition-based maintenance (CBM), lies between these two extremes, wherein maintenance actions are performed as needed, based on the system condition [1]. Ideally, in this strategy, maintenance occurs after a fault but before a failure to reduce any unnecessary maintenance or unplanned downtime[1]. To achieve this objective, this strategy employs predictive analytics over real-time data collected from the system's sensors that detects variation in functional parameters and anomalies that can potentially lead to breakdown [2]. Thus, this strategy can reduce the occurrence of unexpected failures, reduce the machine downtime, reduce maintenance costs, and maximize the equipment life time.

Table 2.1: Pros and cons of each maintenance strategy [2].

Strategy	Advantages	Disadvantages
Corrective	No over-maintenance, overhead of monitoring or planning costs.	High risk of collateral damage and secondary failure and high production downtime, with overtime labor and high cost of spare parts.
Preventive	Maintenance is performed in a controlled manner, with a rough estimate of costs and greater control over spare-parts and inventory. Incurs into fewer catastrophic failures and lesser collateral damage.	Machines are repaired when there are no faults. There will still be unscheduled breakdowns.
Predicted	Equipment life is maximized as unexpected breakdown is reduced or even completely eliminated. Parts are ordered when needed and maintenance performed when convenient.	Can demand higher investment costs. Additional skills might be required.

Table 2.1 summarizes the pros and cons of each maintenance strategy. As sensors and data storage becomes cheaper and more reliable, and equipment are gradually turning to be more complex and requiring higher reliability, the CBM approach becomes increasingly more interesting, as it reduces maintenance costs and increases system availability [9].

[1] While a failure could occur without a clear fault, in this work it is assumed that the system behavior changes before a failure and these changes can be observed by the equipment sensors.

This work has the objective of proposing a data-driven general, agnostic CBM system. The data-driven approach was chosen to make the system deployment efficient and as problem-independent as possible, as any analytical model would rely on the problem domain knowledge with most faults and failures modes lacking analytical models. Considering the inherent difficulties found to produce an analytical model and the current availability of data, an empirical approach where a model would be *learned* was preferred. Also, given the multiplicity of possible problems where such system could be employed, the proposed system was decomposed into smaller and interchangeable modules, giving it more flexibility and a greater range of possible uses.

This chapter presents the proposed CBM system, including its architecture and modules. Section 2.2 presents the concept of learning a model from data. The proposed system architecture and modules are described in Section 2.3. Section 2.4 describes the methodology to assess the system and modules performance under the tasks of detecting and identifying possible faults or failure events. Implementation and computational details are discussed on Section 2.5. Section 2.6 concludes this chapter with a brief summary of its content and introduces the next chapter discussion.

2.2 Learning from data

Consider the problem of describing a phenomenon which lacks an analytic model, but a set of example *cases* or *objects* from the phenomenon universe Ω are available. This universe is composed of an input space $\mathcal{X}$ and an output space $\mathcal{Y}$, with each case or object being a pair $(\mathbf{x}, \mathbf{y})$. By the definition, both the input and the output spaces are assumed to contain all possible input vectors and all possible output values [10], respectively. Thus, our problem becomes finding an optimal empirical model h which maps the inputs to the outputs approximating the target function $f(\mathbf{x}) = \mathbf{y}$ [10, 11]. This requires a cost function $L(\mathbf{y}, h(\mathbf{x}))$ for measuring the errors in this approximation. Thus, by using the loss function as optimization criterion, one can select the optimal model $h^*(\mathbf{x})$. For example, selecting the *mean square error* as losses produces as criterion [12]

$$\begin{aligned} E\left\{L\left(\mathbf{y}, h\left(\mathbf{x}\right)\right)\right\} &= E\left\{\left(\mathbf{y} - h\left(\mathbf{x}\right)\right)^2\right\} \\ &= E_X E_{Y|X}\left\{\left[\mathbf{y} - h\left(\mathbf{x}\right)\right]^2 |\mathbf{x}\right\} \end{aligned} \tag{2.1}$$

which is equivalent of solving pointwise for a given $\mathbf{x}_n$:

$$h^*\left(\mathbf{x}\right) = \underset{h(\mathbf{x})}{\arg\min}\, E_{Y|X}\left\{\left[\mathbf{y} - h\left(\mathbf{x}_n\right)\right]^2 |\mathbf{x} = \mathbf{x}_n\right\} \tag{2.2}$$

and the solution reduces to

$$h^*(\mathbf{x}_n) = E(\mathbf{y}|\mathbf{x} = \mathbf{x}_n), \tag{2.3}$$

the conditional expectation. As such, finding the best estimator for the target function $f(\mathbf{x}) = \mathbf{y}$ is equivalent to find the best estimator for the underlying distribution $P(\mathbf{y}|\mathbf{x})$.

A *learning algorithm* is a powerful way of modeling a phenomenon when no analytic model is available. Let us define a *dataset* $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$ as the subset of N independent and identically distributed (*i.i.d.*) sample cases from Ω which we have access. A learning algorithm $\mathcal{L}$ is a procedure which, given a criterion, a dataset $\mathcal{D}$, and a set of possible candidates models (the *hypothesis set* $\mathcal{H}$), chooses the best model as h^* [11]. As an example, $\mathcal{H}$ could be the set of all linear models from which the algorithm would choose the best linear fit to the data.

As such, the chosen model depends of the hypothesis set, the learning algorithm, the dataset, and the nature of the inputs and outputs. The input vectors $\mathbf{x} = [x_1, x_2, \ldots, x_m]$, also known as *samples* or *instances*, are m-dimensional vectors, where each *variable* $x_i \in \mathcal{X}_i$, also known as *attribute* or *feature*, can be *ordered* or *categorical* [10]:

- $x_i \in \mathcal{X}_i$ is *ordered* if $\mathcal{X}_i$ is a *totally ordered set.* In particular, $\mathcal{X}_i$ is said to be *numerical* if $\mathcal{X}_i \in \mathbb{R}$;

- $x_i \in \mathcal{X}_i$ is *categorical* if $\mathcal{X}_i$ is a finite set of values, without any natural order.

An output or *target* $\mathbf{y} \in \mathcal{Y}$ can be a scalar or a vector, ordered or categorical. The nature and knowledge of the output $\mathbf{y}$ defines the learning problem [10]:

- In a *supervised learning* problem, the training data $\mathcal{D}$ is composed of pairwise samples cases $(\mathbf{x}, \mathbf{y})$.

  - A *classification* problem is a supervised learning problem where $\mathcal{Y}$ is a finite set of classes (or labels) denoted $c_1, c_2, \ldots, c_J$. In this case, the function $g : \mathcal{X} \to \mathcal{Y}$ is known as *classifier* or a *classification rule*;

  - Else, if $\mathcal{Y} \subseteq \mathbb{R}$, $g : \mathcal{X} \to \mathcal{Y}$ is a regressor and this problem is known as a *regression* problem;

- In an *unsupervised learning* problem, no output information is provided [11], only the input examples. In that case, we are interested in modeling the underlying structure of the problem;

- Lastly, in a *reinforcement learning* problem, only some $\mathbf{y}$ are known, but there is a *teacher* which grades the system output decisions. This occurs mostly when the system (or agent) outputs are a sequence of actions where the resulting outcome is only known at the very end of the sequence, such as chess moves or moving through a room.

A CBM problem is a great candidate for the learning approach, since most of the current equipment in industry lack an analytic model. Also, due the evolution in microelectronics, most of these equipment are being monitored by a considerable number of sensors, producing very large datasets. This means many learning datasets are available to be used. As such, during this work we followed a data-driven approach for generating the CBM system. The next section describes the proposed system and each of its components.

2.3 Proposed system architecture

A CBM system can be organized in many forms. In this work we used a modular architecture with four blocks, similar to the ones described in [1, 2], as presented in Figure 2.1:

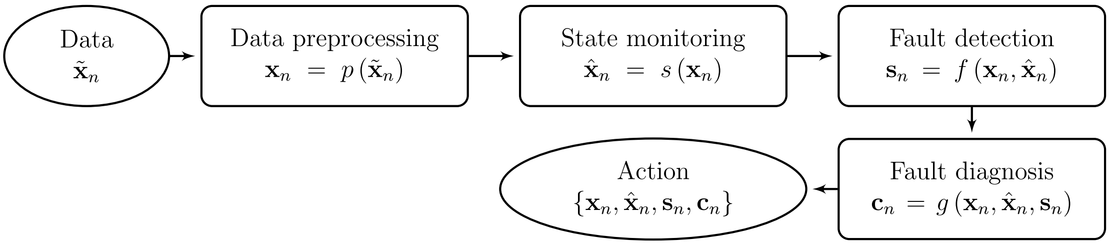

Data $\tilde{\mathbf{x}}_n$
Data preprocessing $\mathbf{x}_n = p(\tilde{\mathbf{x}}_n)$
State monitoring $\hat{\mathbf{x}}_n = s(\mathbf{x}_n)$
Fault detection $\mathbf{s}_n = f(\mathbf{x}_n, \hat{\mathbf{x}}_n)$
Fault diagnosis $\mathbf{c}_n = g(\mathbf{x}_n, \hat{\mathbf{x}}_n, \mathbf{s}_n)$
Action $\{\mathbf{x}_n, \hat{\mathbf{x}}_n, \mathbf{s}_n, \mathbf{c}_n\}$

Figure 2.1: Block diagram of the proposed system.

1. *Data preprocessing*: receives and processes the acquired original data to a more representative format given the application;

2. *State monitoring*: returns the best estimate state of an instance, assuming that the underlying system is at a given behavior. Thus, deviation between the measured and the estimate values can indicate if the system is deviating from a target state;

3. *Fault detection*: evaluates the current instance and detects possible anomalies;

4. *Fault diagnosis*: detects and identifies possible faulty states and locates the corresponding cause;

In short, the proposed system acquires the relevant data. This data is processed (e.g. by moving average, filtering, or linear transformation) and the processed data is transmitted to the monitoring module. This modules estimates the current instance and outputs the residue or similarity between the estimate and the instance. The result is appended to the data, which is evaluated by the detection module. If there is a detection, this information and the current data are used by the diagnosis module which establishes, identifies, and locates possible faults.

The information produced by this system is used to plan and execute maintenance action before the upcoming failure. An example of possible action plan is shown in [2]:

1. Cause analysis given the data;
2. Corrective action planning given the causes;
3. Resource organization given the corrective action;
4. Corrective action implementation.

The next section presents each module in details, including examples and applications.

2.3.1 Data preprocessing

The preprocessing module extracts relevant features from the data to the next modules. This module is context dependent, conditioned by the nature of the data, equipment, failure, and domain knowledge. As an example, frequency domain features can convey more information than the original time series for a rotating machine. In a noise corrupted process, a moving average can be more discriminative than the instantaneous data. Careless data acquisition can pollute the data with redundant or corrupted signals, requiring data cleaning, dimensionality reduction, or attribute selection. The correct preprocessing procedure can improve the system performance as a whole.

2.3.2 State monitoring

This module produces an estimate of the current state considering a possible system condition. This estimate $\hat{\mathbf{x}}_n$ is defined as

$$\hat{\mathbf{x}}_n = s\left(n, \mathbf{x}_{0:n}, \mathbf{v}_n, \mathbf{w}_n\right), \tag{2.4}$$

where $\mathbf{x}_{0:n} = \{\mathbf{x}_0, \ldots, \mathbf{x}_n\}$ is the measured state trajectory from the initial state $\mathbf{x}_0$ to the state $\mathbf{x}_n$ at instant n; $\mathbf{v}_n$ is the observation noise; and $\mathbf{w}_n$ is the process noise[2]. The monitoring function $s(\cdot)$ can be any physical, empirical, or statistical model which represents the target behavior of the modeled system. Ideally, we would have $\hat{\mathbf{x}}_n = \mathbf{x}_n$ for the target mode of operation. As such, the residual or deviation from this behavior can be defined as

$$\mathbf{r}_n = \hat{\mathbf{x}}_n - \mathbf{x}_n. \tag{2.5}$$

The relationship between the estimate and the current state can be transformed into a similarity score r_n by extracting some norm of the residue $\mathbf{r}_n$. A *similarity score* is the output of a *similarity function* which maps the distance between two states into a score $r_n \in [r_{\min}, r_{\max}]$, where an $r_{\min}$ score means no similarity and $r_{\max}$ means identical states. All the similarity functions used in this work are bounded to the interval $[0, 1]$.

2.3.3 Fault detection

This module is a *binary classifier* which indicates when a possible anomalous event occurs given the current knowledge of the system. A binary classifier is a classifier whose target output has only two classes. This module can be described as

$$s_n = f\left(n, \mathbf{x}_{0:n}, r_{0:n}, \mathbf{v}_n, \mathbf{w}_n\right), \quad s_n \in \{0, 1\}. \tag{2.6}$$

The detection module can be employed as a fast event indicator, triggering the diagnosis module when a positive detection case occurs, or to detect possible novel conditions, which would not be detected by the fault diagnosis module alone. A fault detection module can be useful when the diagnosis procedure is costly or complex, mobilizing the diagnosis block only when its strictly necessary. As with the monitoring block, the employed fault detection models are also presented in Chapter 3.

2.3.4 Fault diagnosis

Machine fault diagnosis is the process of classifying features in fault or failure categories. Diagnostics deals with fault detection, isolation, and identification when it occurs [9]. The diagnosis modules is one of the main parts of a CBM system. Not only it supplies information about the fault nature, it also permits to choose the best prognosis model for the given fault [1] and, when the system is already under a failure, to establish the failure causes.

[2] *Process noise* is the inherent noise in the underlying observed process, caused either by changes in the system behavior or our limited modeling, while *observation or measurement noise* is the noise found in the sensors output.

Given the architecture of the CBM system, in the next section we define the methodology to assess and choose the system components.

2.4 Assessment methodology

Considering the modular composition of the proposed system, different models using distinct learning algorithms could be applied for the same purpose. The choice of a model is determined by many factors, including the restrictions in computational cost or complexity; the desired degree of interpretability on the model's decisions; the nature of the features, such as if they are categorical or numerical; the nature of the approached problem; but, between all the possible models for a task, the most important characteristic is the capacity of generalizing to new samples. This *generalization performance* is largely dependent of the specific nature of the problem at hand. Therefore, the assessment is important as it produces the means of selecting the most suitable model and evaluate its quality [12, 13]. This evaluation is not only useful to select the best model h from multiple sets $\mathcal{H}_i$ or multiple learning algorithms $\mathcal{L}_j$, but also to find the best set of parameters for a given pair $(\mathcal{H}_i, \mathcal{L}_j)$ and reducing the risk of *overfitting* the design, leading to inadequate generalization levels [11, 13].

This section presents the assessment methodology used during this work. The modules to be evaluated are the monitoring, fault detection, and state diagnosis modules. The remaining modules are assessed based on their influence on the system performance. To assess the modules performance, first we need some *performance metrics*. These metrics measures the performance of a model given its task. Section 2.4.1 presents some evaluation metrics used for different tasks, such as *classification* and *regression* metrics. Then, Section 2.4.2 presents the procedures used to tune the model parameters, to assess the models, and to select the best model for each task.

2.4.1 Evaluation metrics

Returning to the discussion present in Section 2.2, a learning algorithm is a procedure which given a criterion, a dataset $\mathcal{D}$, and the hypothesis set $\mathcal{H}$, chooses the best model as h^*. An *evaluation metric* is a measure of the *empirical risk* $I_{\text{emp}}[h]$ over a set of samples given a *cost function* $L(\mathbf{y}_n, h(\mathbf{x}_n))$ [14, 15]:

$$I_{\text{emp}}[h] = \sum_i L(\mathbf{y}_i, h(\mathbf{x}_i)) . \tag{2.7}$$

Thus, we can define the learning algorithm $\mathcal{L}$ as the procedure which, based on an evaluation metric, finds the solution h^* which

$$h^* = \underset{h \in \mathcal{H}}{\arg\min} \left\{ \left[\sum_i L\left(\mathbf{y}_i, h\left(\mathbf{x}_i\right)\right) \right] + \lambda r\left[h\right] \right\}, \tag{2.8}$$

where $r\left(h\right)$ is a term which penalizes the complexity of the solution h. As such, the learning algorithm finds a different solution as the evaluation metric changes. For each task, such as classification, regression, or clustering, different metrics are employed. This section presents the metrics used to compare and select models or model parameters for the relevant tasks.

Classification metrics

This section presents some of the possible metrics that can be used to select or assess models for a classification task.

- **Confusion matrix:** also known as contingency table [16], it is not a classification metric per se, but many metrics are derived from it. Each of its columns represent instances in a predicted class while each of its rows represent the instances in an actual class. Thus, the numbers on the diagonal are the number of corrected classified instances for each class. An example of confusion matrix is presented in Table 2.2, where we have three classes of impairment in a rotation machine, 'normal' (without fault), 'imbalance', and 'misalignment'. Each class has 100 instances. Given the confusion matrix $\mathbf{C}$, we can see that the system classifies correctly almost all the 'misalignment' and 'imbalance' fault instances. However, the system fails in discriminating 'normal' samples from other classes. This matrix makes the system shortcomings clear, as it is easy to visually inspect the table for errors, represented by values outside its main diagonal.

Table 2.2: Confusion matrix example.

		Predicted		
		Normal	Imbalance	Misalignment
Real class	Normal	25	50	25
	Imbalance	00	91	09
	Misalignment	00	04	96

- **Accuracy:** or ACC [16, 17], is the proportion of correct guessed instances. Given the confusion matrix $\mathbf{C}$, the accuracy is computed as

$$\text{ACC} = \frac{\sum_i c_{ii}}{\sum_j \sum_k c_{jk}}, \tag{2.9}$$

where c_{jk} represents the entry of the j-th row and the k-th column of the confusion matrix.

For the previous example in Table 2.2, we have ACC = 0.7067, which is reasonable, given the misclassified instances in the normal class. However, this metric can produce misleading results in an unbalanced dataset. Returning to the previous example, if there are only 12 instances of the 'normal' class, respecting the same error proportions, we would have 3 correctly classified samples and 9 misclassified samples (3 with 'misalignment' and 6 with 'imbalance') leading to ACC = 0.9005. These results suggest that this metric is not sensible to unbalanced datasets.

- **Recall:** or true positive rate (TPR) [16], refers to the ability of correctly classifying instances of a given class. For a class i, the TPR is defined as:

$$\mathrm{TPR}_i = \frac{c_{ii}}{\sum_j c_{ij}}. \tag{2.10}$$

A high recall implies low false-negative errors. The maximum recall occurs when all elements of the target class are correctly classified. This metric alone is not enough to evaluate a system, as classifying all instances as the target class would achieve maximum recall.

While the accuracy is a global measure, the recall is a *binary classification* metric which measures the ratio of correctly retrieved instances from a target class label against all the instances with the same label. Thus, one can compute a recall measure for each class, which makes this metric more robust than accuracy against unbalanced datasets.

- **Precision:** or positive predictive value (PPV) [16], measures the number of correctly classified instances from all samples classified as the target class. For the target class i, the precision is defined as:

$$\mathrm{PPV}_i = \frac{c_{ii}}{\sum_j c_{ji}}. \tag{2.11}$$

Precision can be interpreted as the probability that a randomly selected sample predicted as belonging to a target class is a true positive. As such, a high precision implies low false-positive errors. Also a binary classification metric, this metric is often used with the recall metric.

- **F-score:** or F_1 score [16], measures the test accuracy considering both the precision and the recall. The traditional F-score is the harmonic mean of the precision and recall:

$$F_1^i = 2\frac{\mathrm{TPR}_i \cdot \mathrm{PPV}_i}{\mathrm{TPR}_i + \mathrm{PPV}_i}, \tag{2.12}$$

where an F_1 score achieves its best value at 1 and worst at 0.

The diagnosis and evaluation modules are evaluated given these metrics. Also, given the dual nature of the monitoring module, in some cases it can be used as a classifier. In such cases, the module is also assessed by these metrics.

Regression metrics

This section presents some of the possible metrics that can be used to evaluate models for a regression task.

- The **mean absolute error** (MAE) measures how close predictions are from the observed outcome. Given the target outcome y and the predicted value $\hat{y}$, the mean absolute error is given by

  $$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n} \left| y_i - \hat{y}_i \right|. \tag{2.13}$$

  Its low complexity, its relationship to the Minkoski distance metrics [13], and the characteristic of penalizing all errors equally [18] make this metric regularly employed for different problems.

- The **mean squared error** (MSE) is the most famous Minkoski metric [12, 18], being used as a standard metric to measure model performance in multiple areas. Defined as

  $$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n} \left\| y_i - \hat{y}_i \right\|^2, \tag{2.14}$$

  the MSE gives more weight to errors with larger absolute values. Minimizing the MSE produces the optimal estimator when the error distribution is Gaussian [18].

  Lastly, considering a deterministic problem where we have a realization of the training set $\mathcal{D}$, the expected MSE for this problem, independently of any particular realization of the data set, is [11, 12]

$$\begin{aligned}
\mathrm{MSE} &= E_{\mathcal{D}}\left\{\|y_i - \hat{y}_i\|^2\right\} \\
&= E_{\mathcal{D}}\left\{y_i^2 + \hat{y}_i^2 - 2y_i\hat{y}_i\right\} \\
&= y_i^2 - 2E_{\mathcal{D}}\left\{\hat{y}_i\right\}y_i + E_{\mathcal{D}}\left\{\hat{y}_i^2\right\} \\
&= y_i^2 - 2E_{\mathcal{D}}\left\{\hat{y}_i\right\}y_i + E_{\mathcal{D}}\left\{\hat{y}_i\right\}^2 - E_{\mathcal{D}}\left\{\hat{y}_i\right\}^2 + E_{\mathcal{D}}\left\{\hat{y}_i^2\right\} \\
&= \left(y_i - E_{\mathcal{D}}\left\{\hat{y}_i\right\}\right)^2 + E_{\mathcal{D}}\left\{\left(\hat{y}_i - E_{\mathcal{D}}\left\{\hat{y}_i\right\}\right)^2\right\} \\
&= \mathrm{bias}^2\left(\hat{y}_i\right) + \mathrm{var}\left(\hat{y}_i\right). \qquad (2.15)
\end{aligned}$$

We decomposed the MSE in two components: *squared bias* and *variance.* The first term measures how much our learning model is biased away from the target function and is only limited by the learning model itself [11]. The second term measures the 'instability' in the learning problem produced by the variations of the training data, resulting in vastly different learning hypotheses [11]. More powerful models can reduce the bias, as they are free to explore a larger hypothesis set $\mathcal{H}$ for the target function f. However, this comes at the price of possible higher variance, as they have more freedom to learn the idiosyncrasies of the data set, which could reduce their capacity of generalization, leading to overfitting. Conversely, simpler models can reduce the variance at the expense of a small increase in bias. Some training strategies, such as validation, bagging, and regularization, can be used to mitigate the effects of this *bias-variance trade-off* [11, 12].

- The **coefficient of determination or** R^2 **score** measures the proportion of variance in the dependent variable y that is linearly predictable from the independent variables $\mathbf{x} = \{x_1, x_2, \ldots, x_m\}$ [19]. It is computed as

$$R^2 = 1 - \frac{\sum_i^n \left(y_i - \hat{y}_i\right)^2}{\sum_i^n \left(y_i - \mu_y\right)^2}, \qquad (2.16)$$

where

$$\mu_y = \frac{1}{n}\sum_{i=1}^{n} y_i \qquad (2.17)$$

is the dependent variable y mean.

The R^2 score is limited between $[-\infty, 1]$, where 1 means that the dependent variable is completely predicted from the independent variables, whereas a value of R^2 equal or less than 0 means that no prediction is possible with the chosen model or with the current set of independent variables, since this would mean that the results produced by the model are worse than using the mean value of the dependent variable μ_y as predicted value for all instances.

This measure can also be expressed as

$$R^2 = 1 - \frac{\mathrm{MSE}}{\sigma_y^2}, \tag{2.18}$$

which makes this metric a powerful relative measure of the explanation power of the chosen model, as it permits to compare multiple models on different datasets with distinct degrees of "difficulty" (variance).

- The **explained variance regression** (EVR) score is very similar with the R^2 score, as it measures the explanation power of the model given the dispersion of a given dataset. It is defined as

  $$\mathrm{EVR} = 1 - \frac{\mathrm{E}\left\{(e_i - \mu_e)^2\right\}}{\mathrm{E}\left\{(y - \mu_y)^2\right\}} \tag{2.19}$$

  where $e_i = y_i - \hat{y}_i$ and

  $$\mu_e = \frac{1}{n}\sum_{i=1}^{n} e_i \tag{2.20}$$

  is the expected value of the error e. When $\mu_e = 0$, the EVR and the R^2 score are equivalent.

2.4.2 Model selection and assessment

The generalization performance of a learning method relates to its prediction capability on independent data [12]. As such, an accurate estimate of this performance is necessary as a measure of how well the system is expected to perform when deployed [11, 12].

However, finding this estimate is no simple task. One can use the training samples for evaluation, but this strategy disguises the phenomenon of overfitting, producing an overoptimistic result [12, 13]. An alternative approach is to estimate the generalization performance by using a *test set*, a dataset sampled from the same distribution that was not involved in the training process.

While we are very interested in finding the model with the best generalization performance, typically each model will have parameters that need some tuning. If we use a test set to make this tuning, it is no longer a test set [11]. A *validation set* is a subset of the dataset which will not be directly used for training, but is used to make certain choices during the learning process.

Summarizing, we have two main goals during the assessment [12]: *model selection*, where we estimate the performance of different models and parameters in order to choose the best one; and *model assessment*, where, having chosen a final model, we estimate its generalization error on new data. To achieve this goal we would randomly divide the dataset in three parts: a training set, a validation set, and a test set.

While there are multiple methods to achieve these objectives, the simplest and most widely used method for estimating a model performance is *cross-validation* [12]. Among the cross-validation strategies, during this work was employed the one named as K-fold, where the training dataset is divided in K roughly equal-sized subsets [12, 13]. The kth subset is used as a validation set and the remaining $K-1$ subsets are used to fit the model. This procedure is repeated K times for $k = \{1, \dots, K\}$. The cross-validation estimate of the model performance is measure by the mean value of the chosen performance metric between folds. This procedure is illustrated in Figure 2.2 for $K = 10$.

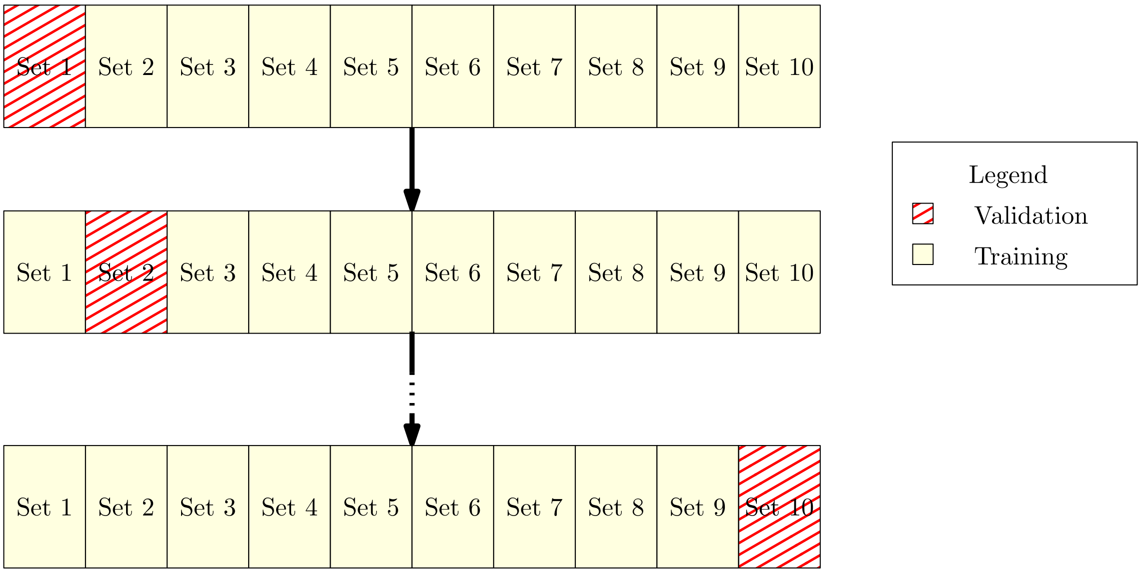


Figure 2.2: K-fold example with $K = 10$.

Typically $K = 5$ or 10, with the case $K = N - 1$, where N is the number of samples, is known as *leave-one-out* cross-validation [12]. This procedure is used to select the best set of models or parameters for a given training set. After all decisions are made, all the training set is used to fit the final model. Then this model is assessed using the test set, thus achieving both of the previous goals.

2.5 Implementation details

This section presents the implementation decisions during this work, including contributions such as the *prognosis* library, which was produced to cope with the lack of known methodologies for monitoring and detecting faults, and the study cases and their respective datasets, which were chosen considering their relevant characteristic to assess the proposed system.

2.5.1 Prognosis library

During this work many algorithms and models where implemented and evaluated. Most of them were deployed in a library named *prognosis*, one of the contributions of this work. The models contained on this library were based on the models founded on API of the *Scikit-Learn* machine learning library [20].

Most of the implemented algorithms used the *Scikit-Learn* library and the numeric and scientific packages *Numpy* [21] and *Scipy* [22].

Currently, the library is composed of 7 main modules: *cbm*, *detector*, *diagnosis*, *monitor*, *process*, *prognosis*, and *utils*. The *process* module contains some of the common preprocessing method employed during this work. The *monitor* and *detector* modules include the models used at the monitoring and detection system blocks, respectively, which are described in Chapter 3. These models follow the estimators from the *Scikit-Learn* library, inheriting from its base estimators.

In the first step of this work, similarity methods were used in the *diagnosis* module, in a stand-alone manner or as an auxiliary module for machine learning models from the *Scikit-Learn* library. The models used for diagnosis are presented during each implementation along the text. The *prognosis* module is discussed in Chapter 7 as a possible future work.

Lastly, the *utils* module contains auxiliary methods and functions, while the *cbm* module contains a container-like class which aggregates all the models as the CBM system depicted in Figure 2.1.

2.5.2 Study cases

This section briefly introduces the study cases and the respective datasets used during this work. Four datasets were employed in this work, ordered in increasing complexity. Each dataset was acquired from distinct sources, with different attributes and characteristics. The first set is an artificial dataset produced to assess the monitoring and diagnosis techniques in a controlled environment. The second and third set are rotating machine datasets, and are used to assess the fault diagnosis system when employed in a static environment, where the system is either healthy or in a faulty state. The last dataset, which contains data from an oil platform injection system, is composed of real equipment data acquired from multiple sensors. As such, most of the problems which can occur during deployment are assessed in this set, such as missing or invalid data, unscheduled shutdowns, redundant attributes, and others.

The next sections describe each dataset excluding the artificial dataset, which was used only for fast evaluation of implementation hypotheses. While each section just introduces each remaining dataset, a detailed description of each set is presented during the course of this work.

Rotating machines datasets

Rotating machines are among one of the main equipment found in the industry, being a key element in a variety of contexts and applications [23, 24]. Two databases were selected to evaluate the system performance. The first one is the machinery fault database (MaFaulDa) [5, 25, 26]. This database includes multiple types and degrees of fault under different conditions. The second database is the Case Western Reserve University bearing dataset [6], a reference dataset in bearing faults, allowing comparison with previous works [27, 28]. These databases are used to assess the proposed system performance and are discussed in the study case presented on Chapter 4.

Oil platform pumps system dataset

This database is related with one of the main aims of this work. A real data problem, it consists of multivariate time-series with attributes of different nature taken from four injections pumps of an oil platform. This dataset collection and preprocessing represent challenges in themselves. First, the sheer size of the database, with each multivariate time-series from each pump sampled minutely during a interval of more than a year. Then, the dataset cleaning procedure, as instances could contain outliers, missing or invalid values. After, the data selection procedure, as many attributes are irrelevant or redundant. Lastly, the weak labels, as there was not a clear indicator of fault or failure, made the data preparation and collection a challenge in itself.

However, this database permits to assess a full system, as it has not only the dynamic which leads to a fault or a failure, but also multiple interactions with the system, such as shutdowns and maintenance stops, and components with distinct behavior, as some pumps where selected as backup and performed only when a primary pump was inoperative. These characteristics make this dataset a “fertile ground” for studying the proposed CBM system. Chapter 5 presents this study case.

2.6 Conclusion

As described in this chapter, condition-based maintenance uses predictive analysis over real-time data to detect possible failure states before their occurrence. As a predictive maintenance approach, it is halfway between corrective and preventive maintenance in maintenance and operating cost.

In this work we propose a modular CBM architecture, based on [1, 2], composed of four modules:

1. *Data preprocessing*, which receives and transforms the original data to a more representative format;

2. *State monitoring*, which estimates the current state given the received data and assuming it comes from a system under normal behavior;

3. *Fault detection*, which detects possible faulty states and returns an indicator;

4. *State diagnosis*, which diagnosis the current or future system state.

To produce an agnostic system which could be deployed at different diagnosis and prediction problems, a methodology based on machine learning was chosen. This methodology permits producing empirical models from pertinent data, such as sensors. Different learning algorithms are evaluated for each module task over distinct study cases dataset, to ensure that the implemented system is the one which achieves best performance. As such, this chapter also presents the methodologies and metrics to measure each component and the system performance.

During the initial exploration of the CBM, we choose to approach this methodology employing *similarity-based methods*. Chapter 3 discusses the motivation behind this decision and introduces the employed methods.

Chapter 3

Similarity-based methods

3.1 Introduction

The most natural way of classifying an instance is by *similarity*. Given a set of examples sharing features and known class labels, we can infer that a new instance shares the same class as the most similar or *nearest* example. The notion of distance or similarity is measured by a *distance* or *similarity metric*, as a representation of 'how far' or 'different' two elements are from each other.

This classifier, which returns as the predicted label of a test instance the label of the nearest or *most similar* instance, is known as the *nearest neighbor classifier.* Its extension, the k-nearest neighbors ($k$nn) classifier, where the test sample is classified by the most frequent label taken from its k nearest neighbors, is one of the most used and known techniques for performing recognition tasks and one of the most interesting algorithms in the data mining field in spite of its simplicity [29]. This approach is so important that in artificial intelligence it is referred to as the instance based learning, memory based learning, or case based learning [30].

However, this approach suffers of some drawbacks [31]. The naive implementation uses all the training samples in order to classify new samples [29, 32]. As the number of samples increases, high storage requirements are necessary, and the computation of similarity between the training and test samples becomes inefficient. Finally, it presents low tolerance to noise as it uses all data as relevant, including noise and invalid data [29, 32].

To tackle some of the drawbacks found on instance learning method, multiple strategies were devised aiming to select a subset of *representative* or *prototype* samples from a large dataset. These approaches are interesting in certain settings were a small number of samples from a large dataset may be of greater interpretative value than generating a model [4]. A subset of these learning methods are the *similarity-based methods*, where a similarity metric is used to compare the new sample against the training prototype set.

As introduced in the previous chapter, the *state monitoring* module produces an estimate for each known condition of the equipment and compares with the current state, producing two types of information: a *residual state*, the pointwise difference between the current state and a estimate; and a *similarity score*, which measures the compatibility between the current state and a given estimate. As an estimate can be computed from the prototype samples, and the similarity score is a natural by-product of the classification procedure, similarity-based methods are a natural choice for this module.

The fault detection module is classifier, or a set of classifiers, which receives the original features and the information produced by the state monitoring module, and returns an indicator of a possible fault or failure system. This module also adds temporal consistency to the monitoring information, as it takes into account the trajectory of the inputs during its decision process. As instance based learning methods are natural classifiers, they can be used as auxiliary modules or they can also act on the detection role.

This chapter presents the proposed approach for the monitoring, detection, and diagnosis systems. Before we can dwell on the secrets of a each of these modules, Section 3.2 describes the *similarity functions*, which are functions that map the distance between two states into a similarity score, the main output of the state monitoring module and the main input of the fault detection module. Each function is presented and a small discussion of their properties follows.

Section 3.3 presents the similarity-based modeling (SBM), the main methodology used in this work and the proposed approaches and modifications over the original method, which are the main contributions of this work. Originally used only in the monitoring module, the proposed modifications transformed the SBM into a multiclass classifier, making feasible its usage in the fault diagnosis module as well.

Section 3.4 introduces the proposed modifications to the standard SBM technique that allow the detection and classification of different states in an efficient and robust manner.

Lastly, Section 3.5 presents a summary of the main points of this chapter and introduces the next chapter.

3.2 Similarity functions

A *similarity function* or *similarity metric* is a function that maps the distance between any two vectors, $\mathbf{x}_i$ and $\mathbf{x}_j$, to a *similarity score* $s_{ij} \in [0, 1]$, such that nonsimilar vectors yield $s \approx 0$ and very similar vectors correspond to $s \approx 1$[1]. A similarity function can be represented as $s(\mathbf{x}_i, \mathbf{x}_j)$, $\mathbf{x}_i \circ \mathbf{x}_j$, or $s(d_{ij})$, where d_{ij} is the distance between any pair of samples $(\mathbf{x}_i, \mathbf{x}_j)$ defined by a distance metric.

A *distance metric* is a function which maps a pair of elements in a set on non-negative real numbers, the distance between them, and satisfies a set of conditions [33]:

$$d(x_i, x_j) \geq 0; \tag{3.1}$$

$$d(x_i, x_j) = 0 \iff x_i = x_j; \tag{3.2}$$

$$d(x_i, x_j) = d(x_j, x_i); \tag{3.3}$$

$$d(x_i, x_k) \leq d(x_i, x_j) + d(x_j, x_k). \tag{3.4}$$

A famous example of distance metric is the family of the p-distance metrics [33], defined as

$$d_{ij} = \left\|\mathbf{x}_i - \mathbf{x}_j\right\|_p = \left(\sum_{k=1}^{m} |x_{ik} - x_{jk}|^p\right)^{\frac{1}{p}}, \tag{3.5}$$

typically used with $p = 2$, known as ℓ_2-norm or Euclidean distance, or $p = 1$, the ℓ_1-norm or Manhattan distance. Another example is the Mahalanobis distance, an extension of the Euclidean distance which takes into account the linear relationships between the variables, defined as

$$d_{ij} = \left\|\mathbf{x}_i - \mathbf{x}_j\right\|_M = \sqrt{\left[\mathbf{x}_i - \mathbf{x}_j\right]^{\mathrm{T}} \boldsymbol{\Sigma}_X^{-1} \left[\mathbf{x}_i - \mathbf{x}_j\right]}, \tag{3.6}$$

where $\boldsymbol{\Sigma}_X^{-1}$ is the dataset covariance matrix. While there are many examples of metrics, during this work we are going to be limited to the p-norms, with $p \in \{1, 2\}$, and the Mahalanobis distance.

In this work, five distinct similarity functions are employed, which can be separated in two main families: the multiquadric set and the exponential set. The multiquadric set include three of the studied functions which are based on the *inverse multiquadric function*, defined as [34]

$$f(d_{ij}) = \frac{1}{\left(r^2 + d_{ij}^2\right)^{\alpha}}. \tag{3.7}$$

The first similarity function is a direct application of this function. By taking

[1] This definition was chosen given that a bounded interval is easier to compare and interpret. In general, any function that returns a real-valued scalar and monotonically quantifies the similarity between two objects is a similarity function. But, since any real-valued function can be mapped in the $[0, 1]$ interval, these definitions are equivalent.

$\alpha = 1/2$, $r^2 = 1/\gamma^2$ and $s_{\mathrm{IMK}}\left(d_{ij}\right) = f\left(d_{ij}\right)/\gamma$, we have

$$s_{\mathrm{IMK}}\left(d_{ij}\right) = \frac{1}{\sqrt{1+\gamma^2 d_{ij}^2}}, \tag{3.8}$$

which is the *inverse multiquadric kernel* (IMK) similarity function.

The second similarity function, the *Cauchy kernel* [35], is a direct variation of the inverse multiquadric kernel similarity function, and can be defined as

$$s_{\mathrm{CCK}}\left(d_{ij}\right) = s_{\mathrm{IMK}}^2\left(d_{ij}\right) = \frac{1}{1+\gamma^2 d_{ij}^2}. \tag{3.9}$$

The last similarity from the multiquadric set is the original similarity function was presented in [36] and used in the original SBM framework, to be discussed on the next section. The *Wegerich similarity function* is defined as

$$s_{\mathrm{WSF}}\left(d_{ij}\right) = \frac{1}{1+\gamma d_{ij}}. \tag{3.10}$$

The last two functions are representative of the exponential set. The *exponential* or *Laplacian kernel* [37] is defined as

$$s_{\mathrm{EXP}}\left(d_{ij}\right) = \mathrm{e}^{-\gamma d_{ij}}, \tag{3.11}$$

and the *radial basis function* (RBF) *kernel* [37] as

$$s_{\mathrm{RBF}}\left(d_{ij}\right) = \mathrm{e}^{-\gamma d_{ij}^2}. \tag{3.12}$$

While originally most of these functions where based on the Euclidean distance, during this work multiple distance metrics are employed to the detection and classification task. The next section introduces state monitoring methodologies and the method which makes use of these functions, the SBM.

3.3 Similarity-based modeling

Similarity-based modeling is a nonparametric modeling technique that uses the similarity of a query vector with exemplar vectors to infer the model's response [38]. This technique was proposed in [3] to monitor and detect faults on a variety of industrial applications. Some of these applications include: fault diagnosis in a machinery fault simulator (MFS) [36, 39]; modeling airplanes flight paths [40]; and anomaly detection in power plants [41]. In the present work the SBM technique was used to monitor the system state and improve the detection and diagnosis procedures performance.

In the original SBM framework, a system state at an instant n can be represented by a vector $\mathbf{x}_n = [x_n(1), x_n(2), \ldots, x_n(m)]^{\mathrm{T}}$ comprising m measures or features from multiple sources, such as system sensors or signals. Given a set of l representative states, selected from a larger set of historical data covering, with minimal redundancy, all the representatives normal states, we can arrange them in a $l \times m$ process "memory" matrix $\mathbf{D}$ as [3]

$$\mathbf{D} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \ldots & \mathbf{x}_l \end{bmatrix}^{\mathrm{T}}. \tag{3.13}$$

Given a state $\mathbf{x}_n$, we can estimate this state as a linear combination of the selected representative states contained on $\mathbf{D}$. To do so, first we define the error between the state $\mathbf{x}_n$ and its estimate $\hat{\mathbf{x}}_n$ as the squared error

$$e_n = \|\mathbf{x}_n - \hat{\mathbf{x}}_n\|_2^2, \tag{3.14}$$

where

$$\hat{\mathbf{x}}_n = \mathbf{D}^{\mathrm{T}}\mathbf{w}_n, \tag{3.15}$$

with $\mathbf{w}_n$ being the optimal linear estimator for a given $(\mathbf{x}_n, \mathbf{D})$ pairs. Then, we can find the optimal linear estimate of $\mathbf{x}_n$ by minimizing e_n

$$\min_{\mathbf{w}_n} \{e_n\} = \min_{\mathbf{w}_n} \left\{ \left[\mathbf{x}_n - \mathbf{D}^{\mathrm{T}}\mathbf{w}_n\right]^{\mathrm{T}} \left[\mathbf{x}_n - \mathbf{D}^{\mathrm{T}}\mathbf{w}_n\right] \right\}. \tag{3.16}$$

Since e_n is differentiable, we have

$$\frac{\partial e_n}{\partial \mathbf{w}_n} = 2\mathbf{D}\mathbf{D}^{\mathrm{T}}\mathbf{w}_n - 2\mathbf{D}\mathbf{x}_n = \mathbf{0} \tag{3.17}$$

which produces

$$\begin{aligned} \mathbf{w}_n &= \left[\mathbf{D}\mathbf{D}^{\mathrm{T}}\right]^{-1} \mathbf{D}\mathbf{x}_n \\ &= \hat{\mathbf{G}}^{-1}\hat{\mathbf{a}}_n, \end{aligned} \tag{3.18}$$

and

$$\hat{\mathbf{x}}_n = \mathbf{D}^{\mathrm{T}} \left[\mathbf{D}\mathbf{D}^{\mathrm{T}}\right]^{-1} \mathbf{D}\mathbf{x}_n, \tag{3.19}$$

where $\hat{\mathbf{G}} = \left[\mathbf{D}\mathbf{D}^{\mathrm{T}}\right]$ and $\hat{\mathbf{a}}_n = \mathbf{D}\mathbf{x}_n$. However, in [3], the authors argue that this result has numerous limitations, such as the requirement that $\mathbf{G}$ must be nonsingular, inability to accommodate random uncertainties and non-random defects, and the need of a very large l. Yet, the relationship between the current measure, its estimate and the system history found on the linear approach has very useful features, such as its simplicity or easy insertion of new data in $\mathbf{D}$.

The SBM was proposed as an alternative approach that copes with these issues and could be used in non-linear systems [3] by substituting the dot product by a similarity function. Following the linear approach, in the SBM framework we can estimate as $\mathbf{x}_n$[2]

$$\hat{\mathbf{x}}_n = \mathbf{D}^{\mathrm{T}} \frac{\mathbf{w}_n}{\|\mathbf{w}_n\|_1}, \tag{3.20}$$

with

$$\mathbf{w}_n = \left[\mathbf{D} \circ \mathbf{D}^{\mathrm{T}}\right]^{-1} \left[\mathbf{D} \circ \mathbf{x}_n\right] = \mathbf{G}^{-1}\mathbf{a}_n, \tag{3.21}$$

that is, $\mathbf{G} = \mathbf{D} \circ \mathbf{D}^{\mathrm{T}}$ and $\mathbf{a}_n = \mathbf{D} \circ \mathbf{x}_n$. The vector $\mathbf{a}_n$ evaluates the similarity between the current state and the representative states on matrix $\mathbf{D}$, whereas matrix $\mathbf{G}$ transforms the similarity vector $\mathbf{a}_n$ in a set of weights for each state in $\mathbf{D}$. When $\mathbf{G} = \mathbf{I}$, the model is called auto-associative kernel regression (AAKR) [42], which works as follows:

- First, compute the distance between the test sample and each prototype;
- This distance is transformed into a similarity score by means of a similarity function;
- These scores are used as weights to compute a weighted mean of the prototypes;
- This weighted mean is the estimated $\hat{\mathbf{x}}_n$.

Thus, the AAKR model can be considered a particular case of the SBM assuming no similarity between the samples within $\mathbf{D}$.

The operator $\circ$ represents a similarity operation having two parameters: a similarity function and a distance metric. It can be represented as

$$\mathbf{x}_i \circ \mathbf{x}_j = s_{\mathrm{F}}\left(\|\mathbf{x}_i - \mathbf{x}_j\|_K\right), \tag{3.22}$$

where s_{F} can be any similarity function, such as the ones described in Section 3.2.

The residual $\mathbf{r}_n$ between the input state and the estimated state, computed as

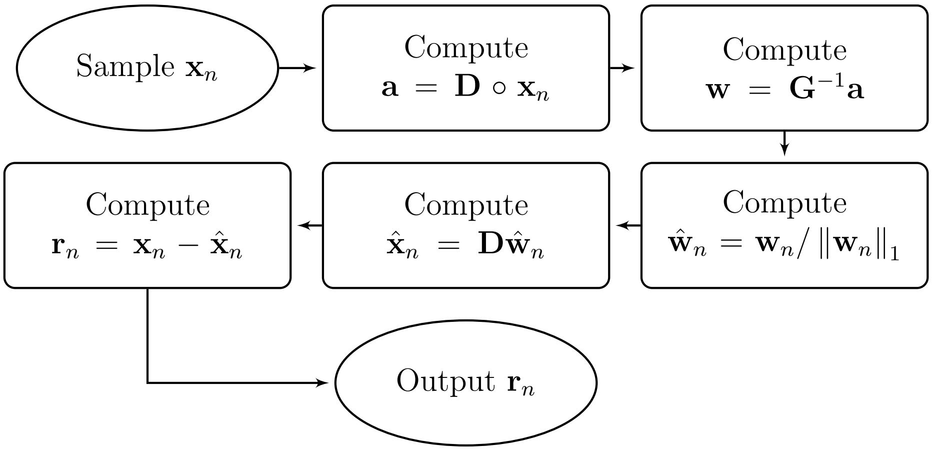
$$\mathbf{r}_n = \mathbf{x}_n - \hat{\mathbf{x}}_n, \tag{3.23}$$

can be used to evaluate and identify the current condition. The residual can be used directly as an auxiliary input for a classifier or the diagnosis system, or it can be employed to compute the similarity score s_n, defined as

$$s_n = s_{\mathrm{F}}\left(\|\mathbf{r}_n\|_K\right). \tag{3.24}$$

[2] While normalizing by $\mathbf{w}_n$ is not required, it is the approach followed in the original SBM formulation. Also, since each term of $\mathbf{w}_n$ is a similarity measured transformed by a semi-positive matrix $\mathbf{G}$, its always positive or null, making this combination convex.

The similarity score s_n can be interpreted as a confidence score in the target state being modeled by the SBM framework. The process of computing the residual $\mathbf{r}_n$ is described in Figure 3.1.



Sample $\mathbf{x}_n$
Compute $\mathbf{a} = \mathbf{D} \circ \mathbf{x}_n$
Compute $\mathbf{w} = \mathbf{G}^{-1}\mathbf{a}$
Compute $\hat{\mathbf{w}}_n = \mathbf{w}_n / \|\mathbf{w}_n\|_1$
Compute $\hat{\mathbf{x}}_n = \mathbf{D}\hat{\mathbf{w}}_n$
Compute $\mathbf{r}_n = \mathbf{x}_n - \hat{\mathbf{x}}_n$
Output $\mathbf{r}_n$

Figure 3.1: SBM sample evaluation procedure.

3.3.1 Original SBM training phase

A key aspect within the SBM formulation is the strategy for composing matrix $\mathbf{D}$. Using all l historical samples for the normal behavior would incur in high computational expenses and redundant data.

High computation expenses as naively computing similarity against each historical sample would incur in $\mathcal{O}(lm)$ complexity; and very redundant data as some states would be overrepresented, making the weights $\mathbf{w}_n$ biases to the most frequent instances and hindering the classification process. However, choosing an inadequate vector set when opting for a smaller l leads to performance impairments. The best possible set, therefore, would have the minimal number of vectors still yielding the same performance level as the complete set. This is always possible since, given a training set $\mathcal{D}$, there is at least one minimal prototype set $\mathcal{P} \subseteq \mathcal{D}$ with the best performance. In [43], a strategy is proposed for selecting a proper reduced set of historical samples. It comprises two selection steps:

1. One chooses as representatives the samples with index in the set $I = \{i_1, i_2, \ldots, i_k\}$, $k \leq 2m$, built such that

$$i \in I \quad \text{if} \quad \exists j : x_{ij} = \min_n \{x_{nj}\} \tag{3.25}$$

or

$$i \in I \quad \text{if} \quad \exists j : x_{ij} = \max_n \{x_{nj}\}. \tag{3.26}$$

2. The remaining samples are selected as prototypes by the following procedure
   (a) First, each sample receives an index, which is sorted by the sample ℓ_2 norm value in decreasing order;
   (b) Then, starting from the first index 0, which represents the sample with the greatest ℓ_2 norm, only the samples with index multiple of t, the decimation factor, are selected to complemented the representative sample set. The remaining samples are discarded.

The first step inserts in $\mathbf{D}$ all vector states which present the minimum and maximum value of each attribute, which could introduce outliers as representative prototypes. The second step decimates the remaining vectors using the ℓ_2 norm as ordering criterion. This second step may also lead to sub-optimal choices, because vectors with similar (even identical) norm values can be completely different [44]. Also, given a small decimation factor t and the number of samples l, the number of chosen prototypes is $\bar{l} = k + \lfloor (l-k)/t \rfloor$, which may be not much lower than l. Additional strategies for composing the matrix $\mathbf{D}$ are proposed in Section 3.4.1 in an attempt to overcome these issues.

3.4 Proposed SBM enhancements

This section presents the proposed enhancements to the SBM formulation, which include: a generalization of the SBM framework that allows it to operate in a multiclass (more than two classes) scenario; introduction of alternative similarity operations; and the development of new strategies to compose the matrix $\mathbf{D}$.

3.4.1 Multiclass similarity-based modeling

The SBM was originally devised to detect abnormal operating conditions, which are associated with a low similarity level between a current state vector $\mathbf{x}_n$ and its SBM estimate $\hat{\mathbf{x}}_n$ given in Eq. (3.20).

Such framework can be extended, however, to detect and classify several types of system operational modes by defining a distinct model-matrix $\mathbf{D}_c$ for each operational class c. In the proposed multiclass SBM formulation, given a new input state $\mathbf{x}$, a different estimate can be determined for each class

$$\hat{\mathbf{x}}_{n,c} = \mathbf{D}_c^{\mathrm{T}} \frac{\mathbf{w}_{n,c}}{\|\mathbf{w}_{n,c}\|_1}, \tag{3.27}$$

where

$$\mathbf{w}_{n,c} = \left(\mathbf{D}_c \circ \mathbf{D}_c^{\mathrm{T}}\right)^{-1} (\mathbf{D}_c \circ \mathbf{x}_{n,c}) = \mathbf{G}_c^{-1} \mathbf{a}_{n,c}. \tag{3.28}$$

The current state $\mathbf{x}$ is then associated to the class c^* which maximizes the similarity score that is,

$$c^* = \arg\max_c \{s_{n,c}\} = \arg\max_c s_{\mathrm{F}} \left(\|\mathbf{x}_n - \hat{\mathbf{x}}_{n,c}\|_K \right), \tag{3.29}$$

where $K \in \{1, 2, M\}$.

3.4.2 Proposed offline training procedure

As discussed in Section 3.3.1, training an SBM model implies choosing a *prototype* set $\mathcal{P}$ from the training data. Choosing the wrong set $\mathcal{P}$ can produce performance impairments. But using all instances as prototypes would incur in high computational expenses, redundant data, and possible overfit. Therefore, the best set would be the smallest set without significantly affecting the model performance.

To meet this aim, many approaches were devised. In [36], for instance, the authors cite a proprietary process which selects the representatives in a one step procedure. Yet, at a previous work ([43]), a two-step algorithm is presented. This procedure consists of two-steps and is described in Section 3.3.1.

However, [44] notices that can exist observations with similar Euclidean norm representing distinct states which would be discarded by this procedure. As such, [44] proposes substituting the second step of the approach proposed in [43] by another approach which divides the feature space in evenly separated bins, adding a new sample if there at least one feature where this sample is near a bin center and its Euclidean distance from any previous selected samples is greater than a given threshold.

Approaches to select prototype instances for classification tasks are very common in the literature of $k$nn classifiers [29, 45]. These methods can be divided into two main approaches: *prototype generation* and *prototype selection* [29]. Prototype generation methods are the ones based on populating the original instance set with optimal representative instances [29]. Most of these methods rely on clustering algorithms to generate the prototype set, such as k-means, k-medoids, hierarchical clustering, among others [29, 45].

Prototype selection methods aim to select the best instances from the dataset which represents or describe the set. This group can divided in three main categories: edition or noise filter methods, condensation methods and hybrid or search methods [29, 46]. Edition methods remove noisy instances to increase the classifier accuracy. Mostly, these points are border points that do not agree with their neighbors. On the other hand, edition methods have the objective of improving the generalization accuracy by producing smoother decision boundaries. In practice, however, these improvements are marginal [29].

Condensation methods start from the whole set, removing redundant samples that do not affect the classifier performance. These methods leave mostly border samples [45, 47]. Some approaches sequentially remove training samples and evaluate the remaining system accuracy. If the accuracy drops below a user-defined threshold, the removed sample is placed on the reference set; otherwise it is eliminated [32].

Hybrid or search methods search for a small subset that simultaneously achieves the removal of both noisy and redundant instances while maintaining or even improving the generalization accuracy [29]. A example of hybrid method is described in [4], where the authors propose a prototype selection method for classification that aims to produce an interpretable set. This method was employed in two of our proposed training procedures and is discussed in this work.

In the present work, alternative approaches were proposed and studied in an attempt to reach the best compromise between the associated computational complexity and the resulting model performance. Three methods were proposed: the *similarity threshold method*, the *interpretable prototype selection method* and the *bootstrapping prototype selection method.*

Similarity threshold method

In this approach, one selects the state vectors $\mathbf{x}_{n,c}$ to form a model-matrix $\mathbf{D}_c$ iteratively. In each iteration a new vector is added to $\mathbf{D}_c$ taking into account the similarity between the currently selected state vectors and the remaining vectors available for class c. More specifically, given a vector set $\mathcal{X}_c$, the iterative procedure starts by selecting its geometric median $\mathbf{v}_c$:

$$\mathbf{v}_c = \underset{\mathbf{z}\in\mathcal{X}_c}{\arg\min} \sum_{\mathbf{x}_i\in\mathcal{X}_c} \|\mathbf{x}_i - \mathbf{z}\|_2 \tag{3.30}$$

as the first representative state to the prototype set $\mathcal{P}_c$ which consists of vectors of $\mathbf{D}_c$. Since there is no closed form to compute this median, it increases in complexity as the number of samples in $\mathcal{X}_c$ increase. In order to reduce the overall complexity, in this work we approximated the median vector by using the algorithm described in [48]. The subsequent states that will compose $\mathbf{D}_c$ are selected according to the following strategy: each new sample $\mathbf{x}_{n,c}$ is compared against the current selected elements in $\mathcal{P}_c$. If the similarity between the $\mathbf{x}_{n,c}$ and any element of $\mathcal{P}_c$ is below a threshold τ, this sample is selected and added to $\mathcal{P}_c$ as an element of $\mathbf{D}_c$, otherwise the sample is discarded. More formally, $\mathbf{x}_{n,c}$ is included in $\mathcal{P}_c$ if

$$\mathbf{x}_{n,c} \circ \mathbf{x}_i < \tau, \quad \forall \mathbf{x}_i \in \mathcal{P}_c. \tag{3.31}$$

This procedure is described in Algorithm 3.1.

Algorithm 3.1 Similarity threshold selection algorithm.

```
function THRESHOLD_SELECTION($\mathcal{X}_c$, $\tau$),
    Compute the geometric median $\mathbf{v}_c = \arg\min_{\mathbf{z}\in\mathcal{X}_c} \sum_i \|\mathbf{x}_i - \mathbf{z}\|_2$
    Initialize $\mathcal{P}_c \leftarrow \{\mathbf{v}_c\}$
    for $\mathbf{x}_{n,c} \in \mathcal{X}_c$ do
        if $\mathbf{x}_{n,c} \circ \mathbf{x}_i < \tau$, $\forall \mathbf{x}_i \in \mathcal{P}_c$ then
            Updates $\mathcal{P}_c \leftarrow \mathcal{P}_c \cup \{\mathbf{x}_i\}$
        end if
    end for
end function
```

Interpretable prototype selection method

This method is based of the prototype selection methods for interpretable classification presented in [4]. As previously described, selecting samples for $\mathbf{D}_c$ is equivalent to select a set of prototypes $\mathcal{P}_c \subseteq \mathcal{X}$. Consider the set of balls with radius ε centered in each point $\mathbf{x}_i \in \mathcal{P}_c$. The best set of prototypes $\mathcal{P}_c$ is a set of balls having the following properties [4]:

Property 1: It should cover as many points from class c as possible;

Property 2: It should cover as few points as possible from other classes;

Property 3: It is sparse. Using as few prototype as possible for a given ε;

This problem can be translated as a *set cover problem.* Given the set of points $\mathcal{X}$, the set cover problems seeks the smallest subcover of $\mathcal{X}$ from the collection of sets that forms a cover of $\mathcal{X}$. If we take $B(\mathbf{x}) = \{\mathbf{x}' \in \mathbb{R}^m : d(\mathbf{x}', \mathbf{x}) < \varepsilon\}$, which denotes the ball with radius $\varepsilon > 0$ centered in $\mathbf{x}$ with distance d from $\mathbf{x}'$. The goal is to find the smallest subset $\mathcal{P} \subseteq \mathcal{X}$, $\mathcal{P} = \bigcup \mathcal{P}_c$, $\forall c$, such that $\{B(\mathbf{x}_i) : \mathbf{x}_i \in \mathcal{P}\}$ covers $\mathcal{X}$.

We can indicate when a instance belongs to the prototype set $\mathcal{P}$ by introducing variables α_j, such

$$\alpha_j = \begin{cases} 1 & \text{if } \mathbf{x}_j \in \mathcal{P}; \\ 0 & \text{otherwise.} \end{cases} \tag{3.32}$$

This problem can be described as [4, 49]

$$\min \sum_{j=1}^{n} \alpha_j \quad \text{s.t} \quad \sum_{j:\mathbf{x}_i \in B(\mathbf{x}_j)} \alpha_j \geq 1 \quad \forall \mathbf{x}_i \in \mathcal{X}, \tag{3.33}$$

where $\alpha_j \in \{0, 1\}$, $\forall \mathbf{x}_j \in \mathcal{X}$.

While Equation (3.33) represents the first property, it does not address the remaining properties. Property 1 states that in certain cases some point from class c should be left uncovered as they would add points with label $y \neq c$. Following [4], we adopt a *prize-collection set cover framework*, assigning a cost to each covering set, penalties for each uncovered or incorrectly covered point, and then find the minimum-cost partial cover [50]. This problem is now described as

$$\min_{\alpha_j^{(c)},\xi_i,\eta_i} \sum_i \xi_i + \sum_i \eta_i + \lambda \sum_{j,c} \alpha_j^{(c)}$$

$$\text{s.t.} \begin{cases} \sum\limits_{j:\mathbf{x}\in B(\mathbf{x}_j)} \alpha_j^{(y_i)} \geq 1 - \xi_i, \quad \forall \mathbf{x}_i \in \mathcal{X}, \\ \sum\limits_{\substack{j:\mathbf{x}\in B(\mathbf{x}_j) \\ c\neq y_i}} \alpha_j^{(c)} \leq \eta_i, \quad \forall \mathbf{x}_i \in \mathcal{X}, \\ \alpha_j^{(c)} \in \{0,1\} \quad \forall j,i \quad \xi_i, \eta_i \geq 0 \quad \forall i, \end{cases} \tag{3.34}$$

where $\alpha_j^{(c)} \in \{0,1\}$ indicates if $\mathbf{x}_j$ belongs to $\mathcal{P}_c$; ξ_i is a slack variable for the Property 1. Thus, ξ_i indicates whether $\mathbf{x}_i$ does not fall within ε of any prototype of class y_i [4]. If a training point from class c is not covered, $\xi_i = 1$; likewise, η_i counts the number of instances with $c \neq y_i$ that are within ε of $\mathbf{x}_i$; finally, $\lambda \geq 0$ is a parameter specifying the cost of adding a prototype [4]. This last parameter is generally set to $\lambda = 1/|\mathcal{X}|$ and used only as a "tie-breaker" between among solutions that do equally well in others properties.

In [4] are presented two approaches for approximately solving this problem: one based on linear programming relaxation with randomized rounding, and the other is a greedy approach. Here we presented the latter, which is used in our prototype selection method, as this also was the preferred solution in [4]. The original implementation is in R, but we reimplemented it in Python.

Problem (3.34) intends to minimize the sum of the number of uncovered points, the number of incorrectly covered points, and the number of prototypes. We can then define a greedy algorithm which finds, at each step, the point $\mathbf{x}_j \in \mathcal{X}$ and class c for which adding $\mathbf{x}_j$ to $\mathcal{P}_c$ produces the maximum cost reduction. The incremental cost reduction can be denoted by

$$\Delta L(\mathbf{x}_j, c) = \Delta\xi(\mathbf{x}_j, c) - \Delta\eta(\mathbf{x}_j, c) - \lambda \tag{3.35}$$

where

$$\begin{aligned} \Delta\xi(\mathbf{x}_j, c) &= \left| \mathcal{X}_c \bigcap \left(B(\mathbf{x}_j) \setminus \bigcup_{\mathbf{x}'_j \in \mathcal{P}_c} B(\mathbf{x}'_j) \right) \right|, \\ \Delta\eta(\mathbf{x}_j, c) &= \left| B(\mathbf{x}_j) \bigcap (\mathcal{X} \setminus \mathcal{X}_c) \right|, \end{aligned} \tag{3.36}$$

such that adding a prototype must add a number of correctly covered samples, $\Delta\xi\left(\mathbf{x}_j, c\right)$, greater than wrong covered samples, $\Delta\eta\left(\mathbf{x}_j, c\right)$, and the cost of a new prototype λ. This procedure is described in Algorithm 3.2, based on [4]

Algorithm 3.2 Interpretable prototype selection algorithm.

```
function PROTOTYPE_SELECTION($\mathcal{X}$, $\mathcal{P}'$, $\tau$),
    if $\mathcal{P}' = \emptyset$ then
        $\mathcal{P}' = \mathcal{X}$
    end if
    Start with $\mathcal{P}_c = \emptyset$ for each class $c$;
    while $\Delta\xi\left(\mathbf{x}, c\right) > 0$ do
        Find $\left(\mathbf{x}^*, c^*\right) = \arg\max_{(\mathbf{x}_j, c)} \Delta L\left(\mathbf{x}_j, c\right), \quad \mathbf{x}_j \in \mathcal{P}'$
        Let $\mathcal{P}_{c^*} \leftarrow \mathcal{P}_{c^*} \cup \{\mathbf{x}^*\}$
    end while
end function
```

Bootstrapping interpretable prototype selection method

Algorithm 3.2 needs a dissimilarity (or distance) matrix as input. If we take a set of N samples as a possible prototype set, the training matrix would be $N \times N$. As N increases, the computational costs become prohibitive. However, if we have a set of prototypes candidates $\mathcal{P}'$, with $P' = |\mathcal{P}'|$ candidates, were $P' \ll N$, we can compute only a $N \times P'$ distance matrix, reducing the algorithm computational cost.

To reduce the computational burden we propose a two-step algorithm using *bootstrapping* to select an initial set of prototypes $\mathcal{P}'$. First, starting with the training set $\mathcal{D}$, we produce K sets $\mathcal{D}'_k$ with N/t samples by sampling $\mathcal{D}$ with replacement, where t is the decimation factor, the same found in the original SBM procedure (Section 3.3.1). Then, we compute Algorithm 3.2 for each set $\mathcal{D}'_k$, producing the prototypes sets $\mathcal{P}'_k \in \{1, \ldots, K\}$. The prototype candidates set $\mathcal{P}' = \bigcup \mathcal{P}'_k$ is then used as initial set for Algorithm 3.2 to select the final prototype set over all the remaining training instances. This procedure is described in Algorithm 3.3.

3.5 Conclusion

This chapter presented the similarity-based methods used in this work for the monitoring, detecting and diagnosis system, including the SBM framework. It also presented some contributions of this work, including new similarity functions, a multiclass extension for the SBM, and new training/prototype selection methods.

The next chapter presents an extension of the SBM framework where the prototypes and their respective classes could be learned on an online setting, which is

Algorithm 3.3 Bootstrapping interpretable prototype selection algorithm.

```
function BOOTSTRAPPING_SELECTION($\mathcal{X}$, $\tau$, $t$)
    $\mathcal{P}' \leftarrow \emptyset$
    for $k = \{1, \ldots, K\}$ do
        $\mathcal{X}'_k$ = BOOTSTRAPPING($\mathcal{X}$, $N/t$)
        $\mathcal{P}'_k$ = PROTOTYPE_SELECTION($\mathcal{X}'_k$, $\tau$)
        $\mathcal{P}' = \mathcal{P}' \cup \mathcal{P}'_k$
    end for
    $\mathcal{P}$ = PROTOTYPE_SELECTION($\mathcal{X}$, $\mathcal{P}'$, $\tau$)
end function
function BOOTSTRAPPING($\mathcal{X}$, $N_B$)
    $N \leftarrow |\mathcal{X}|$
    $\mathcal{X}_B \leftarrow \emptyset$
    for $k = \{1, \ldots, N_B\}$ do
        $r \sim u\,[1, N]$
        $\mathcal{X}_B \leftarrow \mathcal{X}_B \bigcup \{\mathbf{x}_r\}$
    end for
    return $\mathcal{X}_B$
end function
```

useful for real-time applications with an operator giving feedback about the framework decisions.

Chapter 4

Rotating-machines fault diagnosis

4.1 Introduction

Maintenance of critical equipment to ensure high levels of reliability, availability, and performance is one of the major concerns on today's industrial sector [8]. Unexpected failures can lead to substantial losses, either from the maintenance procedure itself or from the resulting production halts [1].

To achieve an effective and cost-efficient procedure, new maintenance strategies are being devised based on real-time and continuous monitoring, allowing one to detect and classify operational anomalies at an early stage, thus limiting additional degradation [1]. Applications of such techniques include, for instance, flight paths [40], natural gas and nuclear power plants [3, 41–43], wind turbines [44], and bearing or rotating-machine faults [5, 24, 26, 36, 38, 39]. Among these equipment, rotating machines are some of the most important, being a key element used in a variety of applications, including airplanes, automobiles, power turbines, oil and gas refineries, and so on [23, 24].

There are many approaches for detecting faults in rotating machines. Most of them consist of extracting features from the vibration signal to assess the machine current condition, in a supervised or automatic manner. Different features are needed to extract useful information relevant to detect faults from the original sources over multiple conditions. These features can be classified considering their domain (time, spatial, time-frequency, frequency) or its computation method (e.g. transform coefficients or aggregated statistics) [27, 51–53].

An illustrative example is the approach in Yang et al. [54]. There, a system is presented which uses an adaptive resonance theory Kohonen neural network (ART-KNN) for fault diagnosis, having as inputs features derived from the discrete Wavelet transform coefficients. Unfortunately, the fault database used is not publicly available, making its comparison with other approaches impractical.

The authors of [55] focus on the feature extraction procedure proposing a novel feature extraction scheme which utilizes the generalized S transform and 2D non-negative matrix factorization to detect possible faults. Three classifiers were used to assess the system: k-nearest neighbors ($k$nn), naive Bayes, and support vector machine (SVM), all achieving good results. A similar approach is presented in [56] using multiscale permutation entropy for feature extraction and a SVM classifier for fault diagnosis. The work of Rauber et. al. [53] also studies the effect of the features in the system performance. It tests multiple features of different types, such as complex envelope spectrum, statistical time- and frequency-domain parameters, as well as wavelet packet analysis, together with a feature selection algorithm. A fault classification database was used as testbed, and three different classifiers ($k$nn, feedforward artificial neural network (ANN), and SVM) were used during the assessment, achieving accuracy above 94%.

This work proposes an automatic fault detector and classifier that uses similarity-based modeling (SBM) to identify rotating-machine failures such as imbalanced load, (horizontal or vertical) shaft misalignment, and bearing defects (in rolling elements or inner/outer tracks). The similarity model can be used either as an auxiliary model to generate features for the classifier (a random forest classifier in this case) or as a standalone classifier. In this context, new approaches for training the similarity model and new similarity metrics are investigated. Two databases were employed to evaluate the performance of the proposed techniques. The first one is the machinery fault database (MaFaulDa) [25], a relatively new, large database of problematic scenarios of rotating-machine operations [5, 26]. Performance evaluation on this database included continuous monitoring of six vibration sensors, one microphone, and one tachometer [5]. The second database is the Case Western Reserve University (CWRU) bearing database [6]. This database has become a standard reference in the bearing diagnostics field [27, 28] and is used as testbed for comparison between the proposed methodology against other algorithms [53, 55–57].

This chapter is organized as follows. First, Section 4.2 details the machinery fault database (MaFaulDa) database, used to design and evaluate the system's performance and the Case Western Reserve University (CWRU) bearing database, used for comparison. Section 4.3 describes the adopted experimental methodology. This section also describes the designed system, including the preprocessing and validation procedures. Section 4.4 discusses the experimental results obtained during the processes of training and selection of the best model, as well the assessment results. Comparisons to other works are also included in this section. Finally, conclusions and discussions emphasizing the main contributions of this work are provided in Section 4.5.

4.2 Databases

Two databases were used to evaluate the contributions of this work. The first one, named machinery fault database (MaFaulDa) [5, 25] is a comprehensive database including multiple types of faults covering different severities and rotation frequencies. This database was extensively used to validate the proposed approach and to select the best models and the best set of parameters based on their performance. The second database is the Case Western Reserve University bearing database [6], the standard reference in bearing faults [27, 28]. It is used to assess the proposed approach against other ones found in the literature. A brief description of each database is presented below.

4.2.1 MaFaulDa

This database is composed of multivariate time-series acquired by sensors on a SpectraQuest's machinery fault simulator (MFS) alignment-balance-vibration trainer (ABVT) [58]. This equipment emulates the dynamic of motors with two shaft-supporting bearings and allows the study of multiple faults, such as imbalanced mass, axis misalignment, and bearing problems. The experimental setup used in this work is illustrated in Figure 4.1.

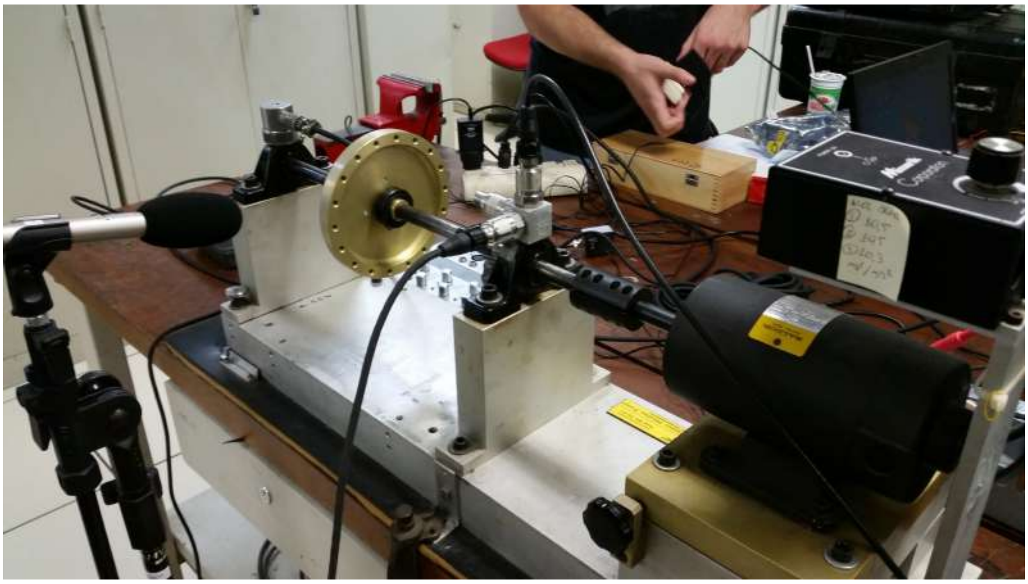


Figure 4.1: Experimental setup used to produce the MaFaulDa database.

The system was monitored by two distinct sets (one for each bearing) of three accelerometers (on the axial, radial, and tangential directions), a tachometer (for measuring the system rotation frequency), and a microphone (for capturing the sound during the system operation). During the signal acquisition procedure, a variety of faults were imposed on the MFS. These faults are described below:

- **Normal operation**: this class represents the system operating under normal condition without any fault. It includes a set of 49 distinct scenarios, each with a fixed rotating speed within the range from 737 rpm to 3686 rpm with steps of approximately 60 rpm.

- **Imbalance**: To simulate different degrees of imbalanced operation, distinct load values of 6 g, 10 g, 15 g, 20 g, 25 g, 30 g, and 35 g were coupled to the rotor. For each load value below 30 g, the rotation frequency assumed in the same 49 values employed in the normal-operation case. For loads equal to or above 30 g, however, the resulting vibration makes impracticable for the system to achieve rotation frequencies above 3300 rpm, limiting the number of distinct rotation frequencies to only 44 in these cases. As such, the database includes a total of 333 different imbalance-operation scenarios.

- **Horizontal Parallel Misalignment**: This type of fault was induced into the MFS by shifting the motor shaft horizontally of 0.5 mm, 1.0 mm, 1.5 mm, and 2.0 mm. Using the same range for the rotation frequency as in the normal operation for each horizontal shift, a total of 197 different scenarios were considered for this class.

- **Vertical Parallel Misalignment**: This fault was induced into the MFS by shifting the motor shaft vertically of 0.51 mm, 0.63 mm, 1.27 mm, 1.4 mm, 1.78 mm, and 1.9 mm. Using the same range for the rotation frequency as in the normal operation for each vertical shift, a total of 301 different scenarios were considered for this fault class.

- **Bearing faults**: As one of the most complex elements of the machine, the rolling bearings are the most susceptible elements to fault occurrence. The ABVT manufacturer provided three defective bearings, each one with a distinct defective element (outer track, rolling elements, and inner track), that were placed one at a time in two different positions in the MFS experimental stand: between the rotor and the motor (underhang position), or in the external position, having the rotor between the bearing and the motor (overhang position). Bearing faults are practically imperceptible when there is no imbalance. So, the three masses of 6 g, 10 g, and 20 g were added to induce a detectable effect, with different rotation frequencies as before, leading to a total of 558 underhang scenarios and 513 overhang scenarios.

Considering all operating conditions described above, the MaFaulDa database comprises a total of 1951 different scenarios, each one described by 8 signals acquired at 50 kHz over a time interval of 5 s. The whole database is available for download at [25].

4.2.2 CWRU bearing database

The data from this database was acquired from the bearing center of the Case Western Reserve University (CWRU) [6]. It consists of 161 scenarios grouped in four categories, as described in [28]. Each scenario can be composed of acceleration signals in three directions: on the drive-end bearing, which occurs in all scenarios; on the fan-end bearing housing, which occurs in most of the scenarios; and on the motor supporting base plate, which occurs in some scenarios. The sample rates used were 12 kHz for some scenarios and 48 kHz for others. The vibration signals were obtained from different states of the bearings: normal condition, inner race fault, ball fault, and outer race fault. A more complete description of this database is found in [6].

The Case Western Reserve University (CWRU) bearing database was selected for two main reasons. The first one is its public availability. The second one is its wide use in the literature for reporting results of automatic bearing fault detection methods, which allows comparison of the performance of proposed method against other works. In this thesis only scenarios containing both the fan-bearing and the drive-end signals were used, reducing the total number of valid scenarios to 153. These scenarios were selected in order to compare against other works in the literature.

4.3 Experimental methodology

This section describes the experimental methodology employed to evaluate the modified SBM performance in detecting and classifying the ABVT's faulty scenarios within the databases described in Section 4.2.

The proposed system follows a modular architecture similar to the ones described in [1, 2] for a condition-based maintenance system. It comprises three blocks (see Figure 4.2): the *preprocessing* module converts the original data to a feature space which is more descriptive for the given application; the SBM model acts as a *state-monitoring* module, returning the similarity between the current input data and the previously modeled conditions; and the *classifier* or *diagnostic* module uses the information from previous blocks to identify the current input among the pre-specified set of classes. In such a framework the SBM can act in a stand alone manner or can be combined to a specific classifier (random forest [10, 59], for instance, as employed here). In this chapter, both strategies are considered.

The preprocessing block has the objective of reducing the original data to a set of more informative, relevant, and less redundant set of values. This is often important for reducing the system burden of learning and generalizing on the original data [11].

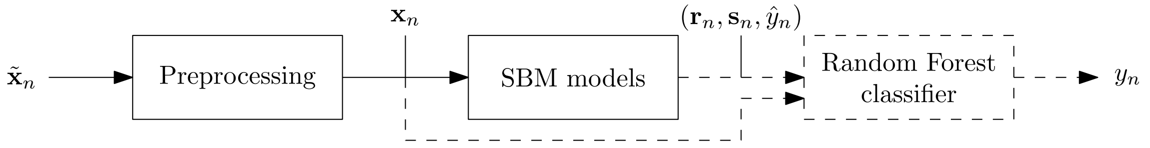


Figure 4.2: Block diagram of the proposed system, composed by a preprocessing module, followed by the SBM and, possibly, by a classifier.

Given the distinct nature of each of the databases employed in this work, the preprocessing block should be different for each database, although its purpose is the same for both. The two preprocessing blocks are described as follows:

(i) *MaFaulDa*: three types of features were extracted from the original multivariate time-series: the *rotation frequency*, 21 additional *spectral features*, and 24 other *statistical features*.

   The rotation frequency f_r was determined directly from the discrete Fourier transform (DFT) of the tachometer signal, as detailed in [23, 26].

   The other spectral features correspond to the magnitudes of the spectrum of the signals other than the tachometer at frequencies f_r, $2f_r$, and $3f_r$.

   The additional statistical features include, for each of the eight measured signals in each operational scenario, the statistical mean, the entropy, and the kurtosis. The variance feature is not employed as the signals are normalized to unit variance to reduce the effect of energy variations caused by changes in the acquisition setup.

(ii) *CWRU*: The statistical features presented in [53], together with the mean, variance and entropy[1] were extracted from each signal, totaling 36 features.

The extracted features are described in Section 4.3.1 The extracted features are then input to the subsequent stages in order to perform fault detection and classification. The two databases are treated independently for performance assessment of the proposed methods. The whole MaFaulDa database was randomly separated in two disjoint training and test sets, comprising respectively 90% and 10% of the given scenarios. The random choice of each set was constrained so that both presented the same fault proportion as the whole database. The best set of parameters was chosen using a k-fold cross-validation procedure on the training samples, with $k = 10$. Then, the performance of the best models are evaluated on the test set, producing the final results shown in Section 4.4.

As for the CWRU database, a process similar to cross dataset validation is applied. The best setups found for the MaFaulDa are directly used on the CWRU

[1] The entropy was computed based on a frequency histogram with a thousand bins.

database. As such, this database is used to assess the generalization power of the classifiers obtained using the proposed methodology. Results have been obtained using k-folds with $k = 10$.

4.3.1 Feature extraction

This section describes the feature extraction procedure for the two databases. In this work three types of features were extracted from the original multivariate time-series: the *rotation frequency*, estimated from the tachometer signal; *spectral features*, extracted from the remaining signals; and *statistical features*, derived from all the sensors signals. These features were based on previous works [5, 26, 53].

Rotation frequency feature

This feature is estimated as described in [26] by computing the N-point discrete Fourier transform (DFT) $S_t(k)$ from the tachometer signal $s_t(n)$, where N is the sequence length in samples. During normal operation, the rotation frequency would be the coefficient with highest energy. However, fault states can introduce spectral peaks with higher energy. To cope with this problem, we followed the two-step estimation algorithm described in [26].

In the first step, an initial frequency estimate is determined as

$$f_i = \frac{k_a F_s}{N}, \quad i \in \{1, 2, 3, 4\} \tag{4.1}$$

where F_s is the sampling frequency and k_a is

$$k_a = \arg\max_k |S_t(k)|. \tag{4.2}$$

After the detection, all frequency coefficients in the range $[k_a - 3, k_a + 3]$ are masked. This procedure is repeated 4 times, generating 4 candidates f_1 to f_4. The final rotating-frequency is given by

$$f_r = \min\{f_1, f_2, f_3, f_4\}. \tag{4.3}$$

Spectral features

These features follows from the rotation frequency, as the machines faults are heavily dependent on the rotation frequency. An N-point DFT is computed for each signal, excluding the tachometer. The features consists of the magnitude of the spectrum at the frequencies f_r, $2f_r$ and $3f_r$.

Since we have 3 features per signal, and 7 signals, excluding the tachometer, this process produces a 21-dimensional feature vector.

Statistical features

These are statistical features computed over the multivariate time-domain series or its spectral transformed version. Some of them were used in previous works on the MaFaulDa database [5, 26], while others were found in [53]. Table 4.1 presents the features, where x_i is the i-th time-domain signal sample, and X_i is the i-th spectral domain coefficient.

Table 4.1: Statistical features taken from time (x_i) and spectral domain data (X_i) from each signal [5, 26, 53].

Time domain	
$\mu_x = \frac{1}{N}\sum_i^N x_i$	$\sigma_x^2 = \frac{1}{N}\sum_i^N (x_i - \mu_x)^2$
$H_x = -\sum_i^N P(x_i)\log P(x_i)$	$\kappa_x = \frac{1}{N}\sum_i^N \left(\frac{x_i-\mu_x}{\sigma_x}\right)^4$
$\gamma_x = \frac{1}{N}\sum_i^N \left(\frac{x_i-\mu_x}{\sigma_x}\right)^3$	$x_{\text{rms}} = \left(\frac{1}{N}\sum_i^N x_i^2\right)^{\frac{1}{2}}$
$x_{\text{sra}} = \left(\frac{1}{N}\sqrt{\lvert x_i\rvert}\right)^2$	$x_{\text{ppv}} = \max_i(x_i) - \min_i(x_i)$
$x_{\text{cf}} = \frac{\max_i(\lvert x_i\rvert)}{x_{\text{rms}}}$	$x_{\text{if}} = \frac{\max_i(\lvert x_i\rvert)}{\frac{1}{N}\sum_i^N \lvert x_i\rvert}$
$x_{\text{mf}} = \frac{\max_i(\lvert x_i\rvert)}{x_{\text{sra}}}$	$x_{\text{kf}} = \frac{\kappa_x}{x_{\text{rms}}^4}$
Spectral domain	
$\mu_X = \frac{1}{N}\sum_i^N X_i$	$X_{\text{rms}} = \left(\frac{1}{N}\sum_i^N X_i^2\right)^{\frac{1}{2}}$
$\sigma_X^2 = \frac{1}{N}\sum_i^N (X_i - \mu_X)^2$	

The feature set from Table 4.1 is divided on 12 time domain statistical features and 3 spectral domain statistical features. The time domain are mean value u_x, variance σ_x, entropy H_x, kurtosis κ_x, skewness γ_x, root mean square value x_{rms}, square root of amplitude x_{sra}, peak-to-peak value x_{ppv}, crest factor x_{cf}, impulse factor x_{if}, margin factor x_{mf}, and kurtosis factor x_{kf}. The spectral domain features are mean value μ_X, root mean square value X_{rms}, and variance σ_X^2. All of these features are used on the CWRU dataset, while some of them, such as time domain mean value, entropy, and kurtosis are used to extended the original feature set described in [5, 26] for the MaFaulDa database.

4.4 Experimental results and discussion

This section describes the experiments made during the validation procedure to select the best model for the proposed task considering all the following system variations:

- Feature types: only spectral features, only statistical features, or both families of features, as discussed in Section 4.3;

- Use of full SBM formulation (as given in Eq. (3.21)) or the AAKR scheme, which considers $\mathbf{G} = \mathbf{I}$ in this same equation;

- Choice of similarity function, as presented in Section 3.2, with distinct values of $\gamma \in \{0.01, 0.1, 0.5, 1, 10\}$ and different ℓ_p norms ($p \in \{1, 2\}$);

- Classification procedure either solely based on the stand-alone SBM or combining it to a specific classifier algorithm (e.g. random forest);

- SBM strategy for building model-matrix $\mathbf{D}$: full matrix with all training feature vectors, standard method [43] (see Chapter 3) for decimation factors $s \in \{2, 3, 5, 7, 11\}$, or proposed threshold-based method (see Section 3.4) for threshold values $\tau \in \{0.05, 0.1, \ldots, 0.95\}$.

Clearly the full combination of the options described above leads to a prohibitively large number of possible system configurations. Therefore, in this work we significantly reduce this number by presenting the validation results following a sequential order of decisions, where each new decision seeks an improvement on the resulting performance. These decisions can be grouped in four main experiments:

- *Experiment 1* evaluates the influence of used feature types;

- *Experiment 2* compares the standard SBM model against the AAKR particularization;

- *Experiment 3* evaluates the different classification strategies, investigating whether one should use the stand-alone SBM to accomplish this function or the SBM output should be fed to a classification algorithm. In latter case, we also investigate the type of feature (similarity value or estimation error vector (Eq. (3.14)) that the SBM module should deliver to the subsequent classifier;

- *Experiment 4* selects the set of the remaining parameters (including procedure for building the model-matrix $\mathbf{D}$ and choice of similarity function) that produces the best SBM model overall.

4.4.1 Validation results

This subsection presents the validation results using cross-validation for each experiment. The experiments are presented in the aforementioned order, where the best configurations found in one experiment are carried out to the next one.

Experiment 1

This experiment assesses the influence of the feature types in the resulting validation performance of the classification system. To reduce the influence of other parameters during this evaluation, a very simple system was used, which employed the SBM as a classifier using the AAKR particularization, and the RBF similarity function with ℓ_2 norm. The kernel width γ assumed all values within the set $\{0.1, 0.5, 1.0\}$. All methods for choosing the SBM prototype matrix $\mathbf{D}$ were assessed, with the decimation parameter fixed at $t = 5$, and the threshold parameter fixed at $\tau = 0.6$.

Table 4.2 presents the obtained cross-validation results for each parameter combination in Experiment 1. From this table, one can readily notice the superior performance achieved by the use of all combined (spectral and statistical) 46 features, which is carried on to all configurations considered in the subsequent experiments.

Table 4.2: Experiment 1 cross-validation accuracy (%), using the AAKR particularization as a classifier with an ℓ_2-norm RBF similarity function. f_r is the rotation frequency.

Prototype selection method	γ	f_r **+ spectral features**	f_r **+ statistical features**	**All features**
Full $\mathbf{D}$	0.1	40.36 ± 3.28	68.27 ± 3.96	$\mathbf{71.61 \pm 3.72}$
	0.5	50.76 ± 3.18	76.55 ± 3.36	$\mathbf{81.00 \pm 3.07}$
	1.0	57.75 ± 3.18	77.0 ± 3.40	$\mathbf{81.91 \pm 2.77}$
Original [43] ($t = 5$)	0.1	37.20 ± 2.10	63.64 ± 3.55	$\mathbf{67.53 \pm 5.04}$
	0.5	46.60 ± 2.48	66.19 ± 4.56	$\mathbf{72.45 \pm 4.69}$
	1.0	49.01 ± 3.21	63.97 ± 3.88	$\mathbf{69.50 \pm 4.15}$
Proposed ($\tau = 0.6$)	0.1	41.02 ± 2.68	61.38 ± 2.78	$\mathbf{67.08 \pm 3.02}$
	0.5	50.36 ± 2.55	76.01 ± 3.28	$\mathbf{80.81 \pm 2.47}$
	1.0	56.70 ± 2.24	77.08 ± 3.69	$\mathbf{81.55 \pm 3.08}$

Experiment 2

This experiment compares the AAKR particularization with the standard SBM model by using the same set of parameters as of Experiment 1. Results from this experiment are summarized in Table 4.3.

From this table, one notices that the standard SBM outperformed its AAKR particularization in all configurations considered here. Given these results, the stan-

Table 4.3: Experiment 2 cross-validation accuracy (%), comparing the SBM and AAKR.

Prototype selection method	γ	**SBM**	**AAKR**
Full **D**	0.1	$\mathbf{84.80 \pm 2.81}$	71.61 ± 3.72
	0.5	$\mathbf{83.66 \pm 2.45}$	81.00 ± 3.07
	1.0	$\mathbf{82.50 \pm 2.39}$	81.91 ± 2.77
Original [43] ($t = 5$)	0.1	$\mathbf{78.75 \pm 3.39}$	67.53 ± 5.04
	0.5	$\mathbf{72.80 \pm 3.26}$	72.45 ± 4.69
	1.0	$\mathbf{70.29 \pm 3.46}$	69.50 ± 4.15
Proposed ($\tau = 0.6$)	0.1	$\mathbf{74.00 \pm 2.28}$	67.08 ± 3.02
	0.5	$\mathbf{82.77 \pm 2.20}$	80.81 ± 2.47
	1.0	$\mathbf{83.01 \pm 2.40}$	81.55 ± 3.08

dard SBM approach was selected as the best performing option to be considered in the experiments that follow.

Experiment 3

This experiment evaluates the SBM method either as a stand-alone classifier or as an auxiliary input to an off-the-shelf random forest (RF) classifier (see Figure 4.2). To this end, we have evaluated four different system configurations: (i) stand-alone SBM classifier; (ii) stand-alone random forest (RF) classifier; (iii) combined SBM-RF classifier using the SBM similarities to each class as a complementary feature; (iv) combined SBM-RF classifier using the SBM estimation error vector defined in Eq. (2.5) as a complementary feature. In this experiment, the SBM model used the best configurations found in the previous experiments, which include all 46 features previously considered and its standard SBM formulation. The RF model used was the one implemented in the *scikit-learn* module [20]. The number of trees estimators was set to 100 and all the remaining parameters were used on their defaults values.

Table 4.4 presents the cross-validation results for all tested SBM-based configurations. As a basis for comparison, note that in our simulations the stand-alone RF configuration achieved an accuracy score of 92.70%.

These results show that the combined SBM-RF schemes are more discriminative than the stand-alone SBM or RF models. We can see that the case where the original features are extended by the similarities to each class estimated by the SBM produces consistently good results. However, the best results were obtained by extending the original features with the SBM estimation error vector (Eq. (2.5)) instead of similarities.

Table 4.4 also indicates that, when one uses the combined SBM-RF configuration with the additional SBM estimation error features, the full model-matrix **D** is

Table 4.4: Experiment 3 cross-validation accuracy (%), comparing different SBM-based classifier configurations.

Prototype selection method	γ	Classifier SBM	 RF + SBM estimation error	 RF + SBM similarities
Full **D**	0.1	84.80 ± 2.81	44.19 ± 4.38	$\mathbf{96.32 \pm 1.66}$
	0.5	83.66 ± 2.45	39.66 ± 3.31	$\mathbf{93.88 \pm 1.88}$
	1.0	82.50 ± 2.39	41.97 ± 3.50	$\mathbf{93.24 \pm 1.39}$
Original [43] ($t = 5$)	0.1	78.75 ± 3.39	$\mathbf{98.20 \pm 1.11}$	96.43 ± 0.89
	0.5	72.80 ± 3.26	$\mathbf{97.66 \pm 0.91}$	94.98 ± 1.25
	1.0	70.29 ± 3.46	$\mathbf{97.44 \pm 0.96}$	94.77 ± 1.21
Proposed ($\tau = 0.6$)	0.1	74.00 ± 2.28	94.66 ± 1.19	$\mathbf{96.59 \pm 1.70}$
	0.5	82.77 ± 2.20	93.87 ± 0.88	$\mathbf{94.64 \pm 1.87}$
	1.0	83.01 ± 2.40	85.92 ± 2.56	$\mathbf{94.02 \pm 1.28}$

greatly outperformed by the ones built using the other two methods. Therefore, in Experiment 4, we consider only the original [43] and proposed methods for building **D**.

Experiment 4

This last experiment performs a fine tuning of the SBM method by selecting the best possible procedure for selecting the prototype matrix **D** together with the best similarity function, including all their parameters.

Table 4.5 presents the 10 best configurations for this experiment, where the 'Parameters' column shows the chosen parameters for each of these scenarios. The similarity functions used are WSF (Eq. (3.10)), RBF (Eq. (3.12)), IMK (Eq. (3.8)), CCK (Eq. (3.9)), and EXP (Eq. (3.11)).

Table 4.5: Experiment 4 cross-validation accuracy (%) for the best 10 SBM configurations. The parameter of the last column is t for the original method for building the model-matrix and τ for the proposed method.

Prototype selection method	Similarity function	Accuracy(%)	Parameters Norm	 γ	 t or τ
Original [43]	WSF	$\mathbf{98.91 \pm 0.58}$	1	0.01	7
	RBF	98.02 ± 0.82	2	0.1	11
	IMK	98.62 ± 0.90	2	0.1	5
	CCK	98.56 ± 0.79	1	1	7
	EXP	98.57 ± 0.93	1	0.1	11
Proposed	WSF	$\mathbf{98.68 \pm 0.89}$	1	0.01	0.9
	RBF	96.66 ± 1.17	2	0.01	0.55
	IMK	98.56 ± 1.37	2	0.1	0.8
	CCK	$\mathbf{98.72 \pm 0.78}$	2	0.1	0.5
	EXP	98.33 ± 1.14	1	0.1	0.5

All the 10 models discriminated in Table 4.5 present superior performance than the one found in previous works using the same database [5, 26], attesting the SBM capability to successfully solve the fault-classification problem in rotating machines.

The analysis of the validation results in Table 4.5 leads us to choose three models as the best ones. They are:

1. Original: Similarity function WSF, $\gamma = 0.01$, ℓ_1 norm. Using the original method for building $\mathbf{D}$ [43], with $t = 7$.

2. Proposed A: Similarity function WSF, $\gamma = 0.01$, ℓ_1 norm. Using the proposed method for building $\mathbf{D}$, with $\tau = 0.9$.

3. Proposed B: Similarity function CCK, $\gamma = 0.1$, ℓ_2 norm. Using the proposed method for building $\mathbf{D}$, with $\tau = 0.5$.

The complexity of a given SBM model is given by its number of representative prototypes. Therefore, in order to analyze the model complexities when using the three above models, we present in Table 4.6 the average number of representative prototypes, over the 10 validation folds, for each combination of selected model and class (imbalanced $\mathbf{I}$, horizontal misalignment $\mathbf{H}_M$, vertical misalignment $\mathbf{V}_M$, underhang faulty bearing $\mathbf{U}_B$, overhang faulty bearing $\mathbf{O}_B$).

Table 4.6: Average number of prototypes for each combination of prototype selection method and class (imbalanced $\mathbf{I}$, horizontal misalignment $\mathbf{H}_M$, vertical misalignment $\mathbf{V}_M$, underhang faulty bearing $\mathbf{U}_B$, overhang faulty bearing $\mathbf{O}_B$).

Configuration	**N**	**I**	$\mathbf{H}_M$	$\mathbf{V}_M$	$\mathbf{U}_B$	$\mathbf{O}_B$
Original	33.1	87.6	65.7	80.1	131.2	118.2
Proposed A	5.8	73.8	8	8	94.8	29.8
Proposed B	5	73.5	4.9	5	74.5	6

From this table, one can readily draw two conclusions regarding the average number of representatives in each case: first, considering each prototype selection scheme, the table shows, as expected, that the proposed threshold method is more selective than the original method. This is an important result, as the proposed method requires less storage space and processing time, making it well suited for deployment in a real-time application. Second, analyzing the size of the model-matrices for each failure, we observe that the imbalance failure and the underhang bearing fault require larger number of states, and are thus difficult to discriminate.

4.4.2 Results on the testing sets

In this subsection the performance of the proposed system on the testing set is analyzed. For this study, the three models chosen in Section 4.4.1 will be considered:

Original, Proposed A and Proposed B. It is important to notice that the original scheme leads to a simpler prototype selection stage but to a larger matrix which results in a more complex classification procedure, as observed in Table 4.6.

As mentioned on Section 4.3, the test dataset is composed by 10% of the available MaFaulDa scenarios. Using the test dataset, the Original and Proposed A models achieved an accuracy of 98.49% and the Proposed B configuration achieved and accuracy of 97.47%, indicating that all three models are capable of generalizing well for other samples. The confusion matrices for the first two models are shown in Tables 4.7a and 4.7b.

Table 4.7: Confusion matrices in test dataset.

(a) Original.

Class	**N**	**I**	$\mathbf{H}_M$	$\mathbf{V}_M$	$\mathbf{U}_B$	$\mathbf{O}_B$
N	4	0	1	0	0	0
I	0	34	0	0	0	0
$\mathbf{H}_M$	0	0	20	0	0	0
$\mathbf{V}_M$	0	0	0	31	0	0
$\mathbf{U}_B$	0	1	0	0	55	0
$\mathbf{O}_B$	0	1	0	0	0	51

(b) Proposed A.

Class	**N**	**I**	$\mathbf{H}_M$	$\mathbf{V}_M$	$\mathbf{U}_B$	$\mathbf{O}_B$
N	5	0	0	0	0	0
I	0	34	0	0	0	0
$\mathbf{H}_M$	0	0	19	1	0	0
$\mathbf{V}_M$	0	0	0	31	0	0
$\mathbf{U}_B$	0	1	0	0	54	1
$\mathbf{O}_B$	0	0	0	0	0	52

Results in these tables are consistent with some already discussed aspects of the MaFaulDa database. As stated in Section 4.2, there are much less scenarios when the machinery operates on normal conditions then there are faulty cases. This discrepancy makes the prototype selection more difficult for the normal class (N). Still analyzing the confusion matrices, it is possible to observe that sometimes bearing faults are misclassified as imbalance faults. We argue that this is somewhat expected, since the system needs to be unbalanced in order for bearing faults to be observed.

Also in Section 4.2, when the bearing faults are described, it was mentioned that each one of the bearings (underhang and overhang) where substituted by one out of three defective bearings provided by the manufacturer. Taking this into consideration, the three best configurations, the Original, Proposed A, and Proposed B, were also used to classify the signals in 10 classes. These classes were derived by further subdividing each bearing fault in 3 classes according to the defective element (outer race, inner race, or rolling ball) employed. The results are presented in Table 4.8, where the good accuracy figures indicate that the proposed system is also robust when applied to more complex fault classification problems.

Table 4.8: Accuracy results for the 10-class identification problem on MaFaulDa.

Model-matrix building method	Similarity function	γ	t or τ	p	Acc. (%)
Original	WSF	0.01	7	2	98.48
Proposed A	WSF	0.01	0.9	1	98.48
Proposed B	CCK	0.1	0.5	2	97.48

4.4.3 Comparison with other prototype selection methods

This subsection presents a comparison of the proposed methods, including the interpretable prototype selection method, described in Section 3.4.2, against some traditional prototype selection methods used on $k$nn-based classifiers: the *condensation* method [60] and the *edition* method [61]. Since in this section we are interested only in comparing different prototype selection strategies, we did not use the RF classifier as an auxiliary classifier for this comparison.

The *condensed nearest neighbor rule* [60] is based on the on the concept of a consistent subset of a dataset. As a lazy learning algorithm, $k$nn imposes high storage cost, as a new sample is classified based on all original samples on a dataset. A consistent subset is a set of samples that correctly classifies all remaining points in the dataset. The condensation nearest neighbor algorithm tries to find the minimal consistent subset of a dataset according to the following steps

1. Store the first sample $\mathbf{x}_1$. Set $i = 2$;

2. The ith sample $\mathbf{x}_i$ is classified using the 1-nn rule, using the current stored samples. If $\mathbf{x}_i$ is correctly classified, store this sample. Move to the next sample;

3. Repeat step 2 until either conditions occurs: all samples are in storage or we passed through all samples not in the storage;

4. Repeat step 3 in the set of samples not in the storage until all samples are in storage or the storage did not change;

5. The stored content is the selected prototype set.

This procedure is very similar with the proposed threshold selection method described in Section 3.4.2, except that the threshold method discards similar samples in the same class, ignoring its effect in classification during its selection procedure.

In another direction, the *edited nearest neighbor rule* [61] starts by storing all samples from the dataset, and then each instance is removed if it is misclassified by its k neighbors (usually with $k = 3$). This procedures removes noise instances and border cases, leaving smoother decision boundaries. The remaining samples are the selected prototype set. As opposed to the condensation algorithm, it also retains internal points, keeping it from reducing storage requirements.

Table 4.9 presents the results of this comparison on the MaFaulDa database. To this comparison, we also used the condensation and edition procedures in the SBM framework as prototype selection methods. As shown, using these prototype selection degrades the performance of $k$nn and SBM algorithms. However, this degradation is more noticeable when using the condensed rule than its edited counterpart.

The interpretable prototype selection method, described in Section 3.4.2, achieves similar accuracy performance as edition. This is very interesting, as these methods differs greatly, with the former selecting internal and boundaries instances and the latter smoothing boundaries by removing boundaries instances. However, their performance are still bellow the proposed heuristics.

Another interesting observation is the degradation in performance of the Proposed B without the auxiliary RF classifier. This difference shows that prototype selection methods could be very sensitive against parameters selection given an application. These observations could be a subject of study of future work.

Table 4.9: Accuracy comparison between $k$nn prototype selection methods against the proposed methods on MaFaulDa.

Model	Accuracy [%]
kNN	92.15 ± 1.45
CNN [60]	74.02 ± 2.47
ENN [61]	90.05 ± 1.24
SBM (Condensation)	90.66 ± 2.00
SBM (Editing)	94.71 ± 1.11
SBM (Interpretable)	94.21 ± 1.77
SBM (Original)	$\mathbf{96.87 \pm 1.08}$
SBM (Proposed A)	$\mathbf{97.95 \pm 0.56}$
SBM (Proposed B)	76.10 ± 3.12

4.4.4 CWRU results and discussion

This subsection presents the results on the CWRU bearing dataset. As described in Section 4.3, this dataset is used for assessing the performance of the three best models selected on the MaFaulDa dataset, namely the Original, Proposed A, and Proposed B schemes (see Section 4.4.1).

Using the same methodology as [53], each CWRU signal was divided into 15 segments, and the extended dataset was subdivided into the training and testing sets following a 9/1 ratio. The results presented in Table 4.10 are accuracy averages over 10 folds chosen randomly. For each fold configuration, the model-matrix **D** is computed using the data in 9 folds and the accuracy result is measured in the remaining fold. From this table, one can observe that the SBM-based classifier has good generalization capability for all three configurations considered here.

Table 4.10: Accuracy (%) results of SBM-based classifiers on the CWRU database.

Model	Accuracy
Original	98.95 ± 0.72
Proposed A	98.91 ± 0.75
Proposed B	98.91 ± 0.95

4.4.5 Comparison with previous works

As mentioned in Section 4.1, several other works in the literature addressed the same problem that we have addressed in this work, that is, the automatic detection and classification of faults in rotating machines. Some of these works have used the MaFaulDa database. In [5] the faults in the MaFaulDa database have been classified using perceptron neural networks with multiple layers, considering several subsets of the features investigated here. Six classes have been considered: normal, overhang and underhang faults, imbalance, horizontal and vertical misalignment. The accuracy obtained was 95.8%, inferior to the ones obtained with the proposed use of SBM and described in Table 4.8, that reach, for one configuration, the average figure of 98.48%.

For the CWRU database, even though there are many works using such dataset [28], it is very difficult to make a direct comparison, as most works do not present their results in a quantitative manner, but only in a qualitative manner. As such, the comparison is restricted to a small set of works. In [55] the $k$nn, naive Bayes, and SVM classifiers achieved accuracies of 98.83%, 98%, and 98.97%, respectively. The SVM classifier found in [56] obtained accuracies above 98% for different rotation frequencies. The SVM and ELM classifiers using the procedure described in [57] achieved accuracies of 82.4% and 97.5%, respectively. Lastly, the $k$nn, SVM, and ANN classifiers using the feature selection method proposed in [53] obtained accuracies between 93% and 100%. From the above results and Table 4.10, one can conclude that the proposed SBM-based fault classifier achieves, for the CWRU database, competitive results to those found in the literature. It is important to point out that, as demonstrated by the results over the MaFaulDa database, the proposed system is able to detect and classify, with high accuracy, a wide range of machine faults, including misalignment and unbalanced faults.

4.5 Conclusion

This chapter addressed the automatic fault diagnosis in rotating machines. The use of similarity based modeling (SBM) was investigated, either as a stand-alone classification method or in combination with an off-the-shelf classifier, in this case a random forest classifier. The system is evaluated in two databases. One of them is a comprehensive database with multiple faults referred to as MaFaulDa [25]. The other is the CWRU bearing database [6], that is the current standard database for bearing fault diagnosis.

One of the main contributions of this work was the extensive study of the use of multiclass SBM on the MaFaulDa database. Other contributions regard a novel method for building the SBM model-matrix and the use of new similarity metrics. These contributions achieved the goal of increasing the SBM performance in a fault classification scenario while reducing its computational complexity. The usage of SBM either as a stand-alone classifier or as a feature generator for off-the-shelf classifiers has also been investigated. Our results have shown that the use of the proposed enhancements to the SBM consistently increased the accuracy of a random forest classifier.

The proposed system showed to be robust, reaching an accuracy of around 98.5% in the MaFaulDa database, higher than previous works along the same base. For the CWRU dataset the proposed system yielded an accuracy level of 98.9%, which is as good as previous results reported in the literature. It is worth emphasizing that the proposed class of methods is able to detect and classify, with high accuracy, a wide range of faults, which indicates that the proposed approach based on SBM is worth further investigation.

Chapter 5

Failure detection in an oil-platform pump system

5.1 Introduction

The optimal maintenance strategy is one which ensures high levels of reliability, availability, and performance [8]. In the oil and gas industry, where the correct performance is crucial and failure could cost millions of dollars [62], the condition-based maintenance approach is highly interesting, even considering the higher costs and required additional skills [2, 9].

This chapter presents a real-life application of the proposed framework, where it is applied to multivariate time-series from an oil platform pumps set. This study case presents all the aspects of a data-driven problem, from data acquisition to training and deploying the final model. This chapter deals with the solution of a real problem in the industry. The current oil platforms have a myriad of sensors measuring relevant characteristics of the equipment. These sensors are evaluated by a human operator which assesses the equipment current state. However, the increasing number of sensors and the current acquisition frequencies produce an amount of information that is too large to be evaluated by a human in real-time. To cope with this problem, the operator relies on ad-hoc metrics used to identify critical events. This approach demands domain-knowledge and produces reactive actions, limiting the possible maintenance strategy.

The proposed system is a possible solution to this problem. It consists of a system capable of detecting possible failure trajectories to a possible critical event, using only historical information from the sensors. Using a minimum of domain-knowledge, the system should be data-agnostic, permitting its use with multiple equipment and processes, assisting the decision of an operator by generating interpretable warnings.

This chapter is organized as follows. Section 5.2 presents the acquired database which was used during this work, including the different attributes of the multivariate time-series. The adopted methodology, including the preprocessing steps and the employed analyses are presented in Section 5.3. This section also includes a small study of the underlying structure of the database, with the objective of extracting knowledge from the data to be used in the detection procedure. The obtained results using the proposed framework for failure detection are presented in Section 5.4. Lastly, Section 5.5 concludes this chapter.

5.2 Database

With the aim of creating empirical models for diagnostic and validate future procedures, a training and evaluation database was necessary. As such, we used a database composed of multivariate time series corresponding to a set of sensors of four injection pumps, numbered 1 to 4. Each sensor is identified with a single tag, which is common across distinct pumps. As such, each pump can be evaluated independently or as realizations of an archetypal pump model.

The multivariate time series comprises the interval between August 01, 2014, and November 01, 2015, being composed of 41 attributes. Each attribute has distinct characteristics, including: measured values, such as temperature, pressure and vibration sensors, or computed values; continuous or discrete values; and numerical or categorical values. Each attribute was acquired with a one minute sampling period and could contain outliers, missing or invalid values. Given these characteristics, the attributes were discriminated in two main groups to be preprocessed and evaluated:

- *Numerical data*: A total of 36 attributes, divided in two groups: *measured* and *computed.* Four computed series were ignored, as they could be derived from the remaining series. The remaining 32 measured series were preprocessed and used as input for the system;

- *Categorical data*: Each of the 5 categorical series corresponds to a state sensor. Therefore, the categorical series were treated as state labels indicating possible faults and failures and separated from the numerical series.

Before these series could be of any use they had to be preprocessed. The outliers, missing and invalid values on the series must be treated. Also, since the categorical data hints possible anomalous states, this information must be treated and aggregated. Section 5.3 describes the employed preprocessing procedure.

5.3 Methodology

This section describes the methodology adopted to evaluate the proposed framework for detecting faults or failures on the oil injection equipment database described in Section 5.2.

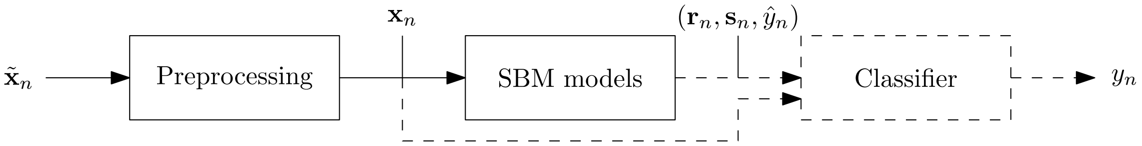


Figure 5.1: Block diagram of the proposed system for fault detection, composed by a preprocessing module, followed by the SBM and, possibly, by a classifier.

The system employed in this problem is very similar to the one presented in Fig. 4.2, reproduced as Fig. 5.1, with the difference that in the current problem we choose to not use any auxiliary classifier in the diagnosis procedure, using the SBM as monitoring, detection and diagnosis method. This decision was made: a) to evaluate the performance of the proposed SBM modifications; b) to reduce the computational burden during the training procedure; and, c) motivated by an implementation issue, as the deployment environment could not accept models generated from the *scikit-learn* package [20], given that the models were trained on 64-bit Windows but the deployment version was 32-bit.

The preprocessing block in this case has another objective that was not discussed in Chapter 4. In an industrial environment with multiple sensors, problems can compromise the reliability of the data, polluting it with redundant or corrupt signals. These "dirty" signals can impair a fault detection system performance. Also, some signals can be presented in formats which are not ready to be used by the proposed system. They need to be converted to a more accessible format first. In this case, the employed preprocessing block must also perform *data cleaning* and *data transformation* on the input signals.

In a data-driven problem, sometimes we have limited or no access to a specialist. In these cases, one must extract all the necessary knowledge to solve the problem at hand directly from the data. *Exploratory data analysis* is a methodology to examine a dataset and obtain knowledge about its underlying structure. This knowledge can reduce the need for a specialist, directing the queries to the relationships found, and to aid the trained models. To assess the findings and the trained models, the database was separated in two sets: equipment 1, 2 and 4 were selected as training set, as equipment 1 had a defect in one of the sensors, and equipment 4 was used as backup, changing its behavior from the expected "normality". Equipment 3 was employed as test set. Since the data from each equipment has the same number of samples, this procedure is equivalent to selecting 75% and 25% of the samples as training and test sets, respectively. This also reduces the chances of data leakage [11].

5.3.1 Data preprocessing

Numerical data series

As previously stated, the numerical attributes were divided in measured and computed attributes. The computed attributes were discarded, as they carried redundant information and would not contribute to the understanding of the system operation modes.

The remaining measured numerical attributes were treated as follows. First, the outliers and invalid values were replaced by missing values indicators. Then, the missing values were filled using linear interpolation. Given a missing value x_{nm} corresponding to the mth attribute of the nth sample in an interval of length $r - l$, where l and r are the indices of the nearest samples with valid values from the left and the right, respectively, we have

$$\begin{aligned} x_{nm} &= \alpha x_{rm} - (1 - \alpha) x_{lm}, \\ \alpha &= \frac{t_n - t_l}{t_r - t_l}, \end{aligned} \tag{5.1}$$

where t_n, t_r, and t_l are, respectively, the time instants of the sample values x_{nm}, x_{rm}, x_{lm}. The interpolated data was used as a input for the next stages.

Categorical data series

Given that the employed methodology treats the diagnostic problem as a supervised learning problem, labels indicating the state of the system for each sample are necessary. Consequently, the categorical attributes were considered as labels indicating the current state. However, it was observed that these attributes were very volatile and redundant. Therefore, without the opinion of a specialist, any conclusion taken on this data would be questionable.

Considering the above, other options were evaluated. The best solution found consists of integrating the information given by the categorical attributes with information extracted from daily reports. These documents contained the daily conditions of a oil platform's section and a succinct report of relevant events. Each equipment state from the given section is recorded and classified in three classes: *operating*, *on standby*, and *inoperative*, with possible overlap between states (e.g. *operating* and *on standby*), as the equipment state could suffer many changes during daily operation. This report is schedule to be produced daily at three different time intervals. However, the studied section had only reports produced between

12:00 a.m. and 07:00 a.m. As such, the information contained in each report consists mostly of the previous day events.

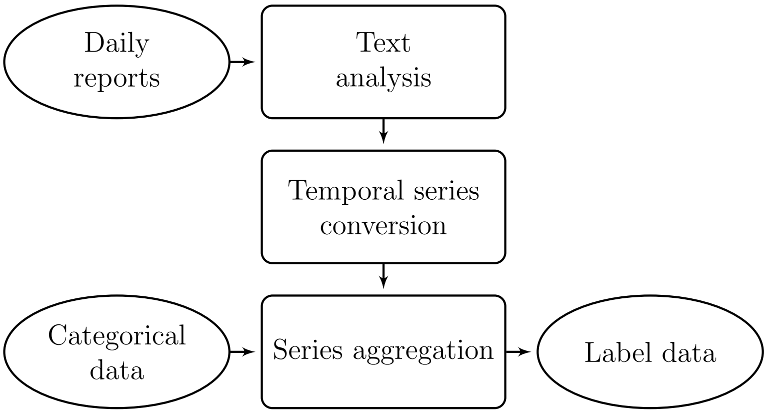


Figure 5.2: State labels generation procedure.

Information concerning the injection pumps was extracted from reports to generate the state labels, following the procedure illustrated in Figure 5.2. The first step was obtaining the daily reports and extracting the status of each pump over time. Almost 459 days of reports, encompassing the interval of 31 July, 2014, and 01 November, 2015, excluding some missing reports, were acquired, read, and manually converted. Each pump state for each day, including overlaps and related events, were annotated. The next step consisted of converting the report events and states to time series with the same period and length as other categorical attributes. As neither the interval which the event occurred nor its start or end are clearly informed, we assumed that the occurrence of the event lasted the length of the report day; as such each minute received the same label. The new categorical series referred as $\{s_n\}$, is aggregated to form the label series. This process is detailed below:

1. Given the report states series $\{s_n\}$, with three possible states (*operating*, *on standby* and *inoperative*), we can define a new binary series where:

$$b_n = \begin{cases} 1, & \text{if } s_n = \textit{operating}; \\ -1, & \text{otherwise.} \end{cases} \tag{5.2}$$

2. Then, each categorical attributes series, $\{c_n^k\}$, $k \in \{1, 2, 3, 4, 5\}$, is aggregated with the binary series $\{b_n\}$ as follow:

$$\hat{y}_n = \begin{cases} b_n, & \text{if } c_n^k = \textit{normal}, \forall k; \\ -1, & \text{otherwise.} \end{cases} \tag{5.3}$$

3. Lastly, two numerical attributes, $\{a_n^k\}$, $k \in \{1, 2\}$, measuring the pump ro-

tation speed and discharge pressure, were used as indicators of the *equipment shutdown.* A shutdown state is assumed when any of these series goes to 0

$$y_n = \begin{cases} \hat{y}_n, & \text{if } a_n^k \neq 0,\ \forall k; \\ 0, & \text{otherwise.} \end{cases} \tag{5.4}$$

4. At the end of this process, we have a ternary time series including all normal states $y_n = 1$, shutdown states $y_n = 0$, and unidentified or anomalous states $y_n = -1$.

This procedure was repeated for each day, except when no report was found. This exception occurred six times. In these days the reports were considered as anomalous and the corresponding samples discarded. In the future, information from other sources could be used to define these states.

5.3.2 Exploratory data analysis

Exploratory data analysis is a methodology to examine a dataset which employs a diversity of techniques, most of them visual methods, to summarize their statistical characteristics, identify important variables, find its underlying structure, and evaluate hypotheses [63].

Exploring the dataset trying to identify possible relationships and behaviors, as well as extracting knowledge, can reduce the demand of domain specialists, limiting their analyses to assess and discard detected insights.

This section presents the analyses used as an attempt to identify the underlying structure which governs the relationships between the different attributes during distinct states of the equipment. To achieve this, we used the training data from different pumps as realizations of the same underlying process, aggregating the samples in a single set. This procedure was done to populate the training set and to produce general assumptions, reducing the danger of just modeling the behavior of a specific piece of equipment.

A way to measure the relationships between the attributes is the *correlation matrix.* The correlation matrix represents the linear dependency between pairs of variables [64]. Before we define the correlation matrix, first we need to define the *covariance matrix.* It is defined by

$$\mathbf{\Sigma}_{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^{N} [\mathbf{x}_n - \boldsymbol{\mu}_x] [\mathbf{x}_n - \boldsymbol{\mu}_x]^{\mathrm{T}}, \tag{5.5}$$

and describes the linear relationship between the attributes [64]. However, it is not a normalized measure of this relationship, preventing comparisons. A correlation matrix can convey this relationship in a normalized fashion. It can be generated from the covariance matrix as

$$\mathbf{R}_x = \operatorname{diag}\left(\mathbf{\Sigma_x}\right)^{-\frac{1}{2}} \mathbf{\Sigma_x} \operatorname{diag}\left(\mathbf{\Sigma_x}\right)^{-\frac{1}{2}} . \tag{5.6}$$

Elements of this matrix can assume values at the interval $[-1, 1]$, where the value 1 represents direct increasing linear relationship (correlation), -1 direct inverse relationship, and 0 indicates no linear relationship. As such, the value of an element r_{ij} from matrix $\mathbf{R}_x$ denotes the degree of linear relationship between the ith and jth variables. Since this is a linear relationship, $r_{ij} = r_{ji}$, and for the special case where $i = j$, $r_{ii} = 1$.

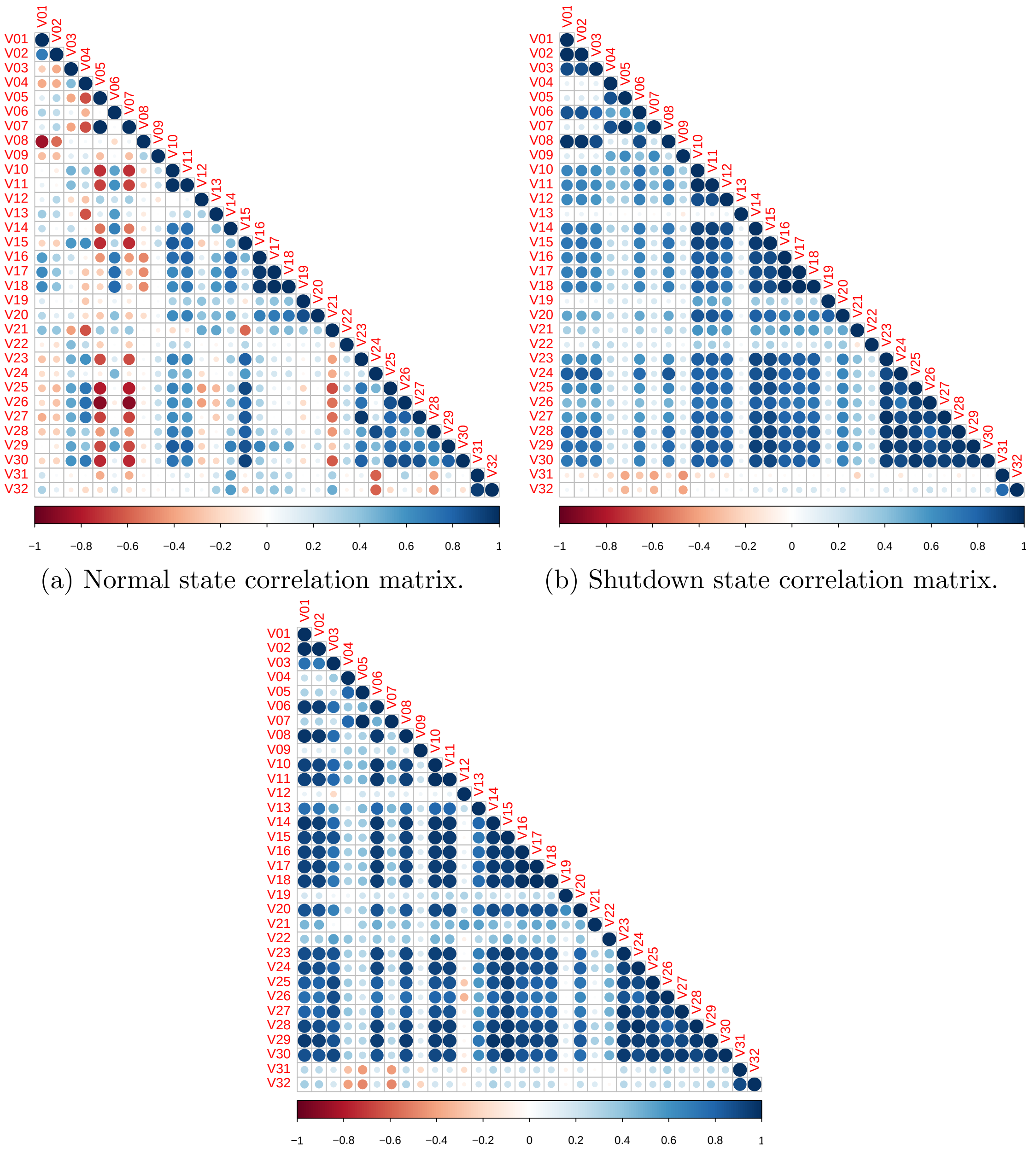
Figures 5.3a, 5.3b, and 5.3c present, respectively, the correlation matrix of the injection pumps attributes under normal, shutdown, and anomalous states. Since correlation matrices are symmetric, only the main diagonal and the lower triangular coefficients are show.

As expected, there are many differences in the relationship between the attributes during the different states. During the shutdown and anomalous states (Figures 5.3b and 5.3c, respectively) most of the attributes are positively correlated, with very strong correlation coefficients. However, a completely distinct pattern is found during the normal state. A greater variety of correlation values occurs, with strong positive or negative correlations occurring in a lesser extent. This behavior indicates that the changes in the relationship between some variables could be indicators of changes in the system state from a normal state to an anomalous or shutdown state. Still, the correlation matrices of the shutdown and anomalous states are very similar. Some possible causes for this behavior are: the shutdown and anomalous states are not linearly separable, projecting into similar correlation matrices; considering that anomalous states could lead to shutdown, there is a natural overlap; or the proposed labels do not correctly describe these states.

With the objective of visualizing and assessing the relationships between the variables, another approach is used for further exploring the correlation matrices. This approach is based on observing the relationships found in the *partial correlation matrix*. The partial correlation matrix is the inverse of the correlation matrix and is computed as

$$\mathbf{P}_x = \mathbf{R}_x^{-1}, \tag{5.7}$$

it measures the degree of association between two variables, its entry (i, j) is the correlation between variables i and j if all others are kept constants [65]. Considering a normal distribution, each zero entry coefficients found on $\mathbf{P}_x$ indicates the conditional independence between pairs of variables. As such, we can learn a graph structure which represents the relationships between the variables from a sparse



(a) Normal state correlation matrix.

(b) Shutdown state correlation matrix.

(c) Anomalous state correlation matrix.

Figure 5.3: Correlation matrices for the three states: normal, shutdown, and anomalous. The color intensity and the circle size represents the coefficient absolute value. A red circle indicates negative correlation; a blue circle indicates positive correlation.

partial correlation matrix. For this aim we followed the approach of [65], where, given a set of attributes $\{x_1, \ldots, x_m\}$, we can produce an estimator $\boldsymbol{\theta}^{i,\lambda}$ for a given x_i and λ, such that

$$\boldsymbol{\theta}^{i,\lambda} = \underset{\boldsymbol{\theta}^i : \theta^i = 0}{\arg\min} \left(\frac{1}{N} \left\| \mathbf{X}_i - \mathbf{X}\boldsymbol{\theta}^i \right\|_2^2 + \lambda \left\| \boldsymbol{\theta}^i \right\|_1 \right), \tag{5.8}$$

where $\mathbf{X}$ is the matrix composed by stacking samples $\mathbf{x}_n$ as

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \ldots & \mathbf{x}_N \end{bmatrix}^{\mathrm{T}}$$

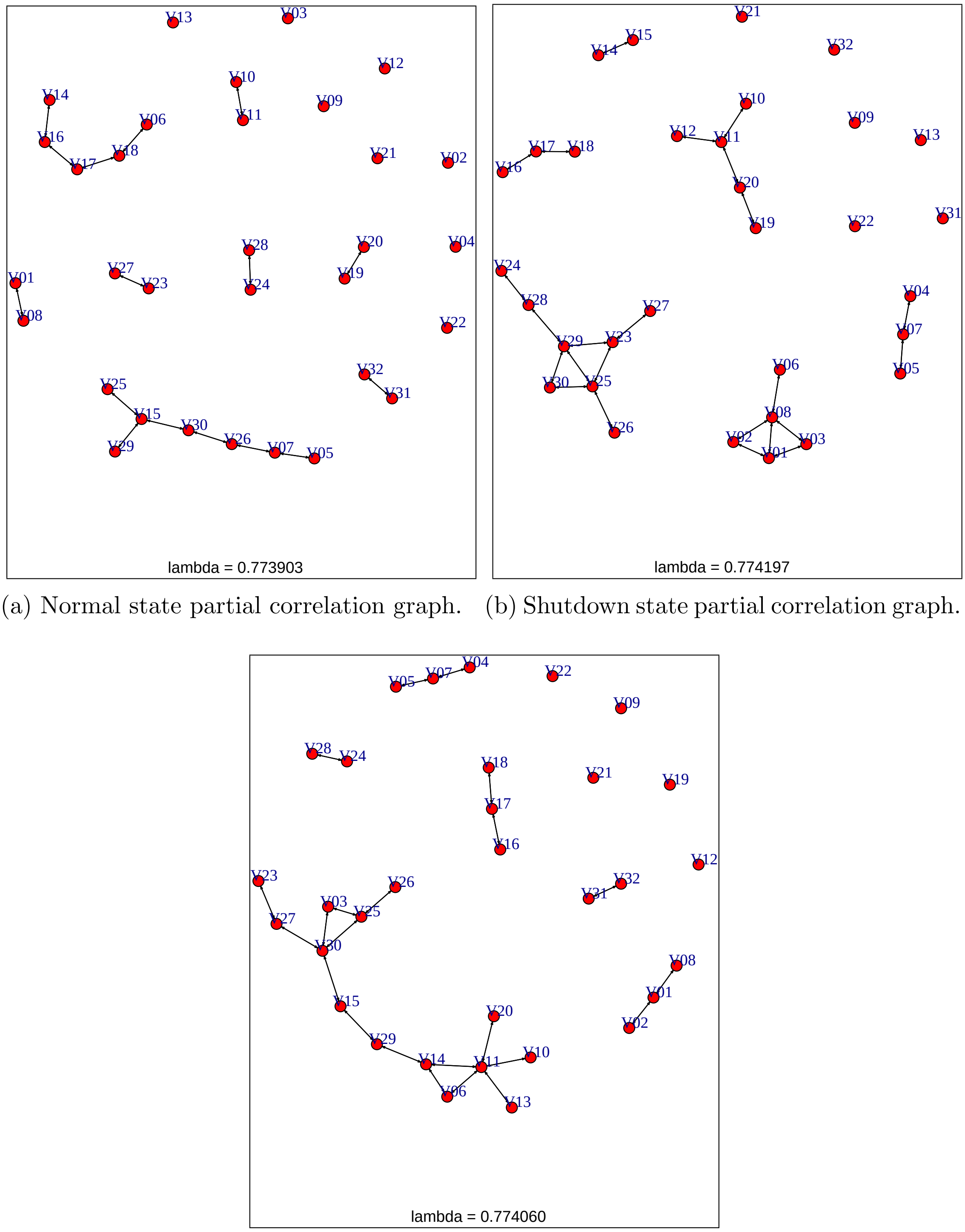
and $\mathbf{X}_i$ is $1 \times N$ feature vector produced by the i-th column of $\mathbf{X}$.

The set of variables x_j such that $\theta_j^{i,\lambda} \neq 0$ is the *neighborhood* of x_i, with x_i being conditionally independent linearly from the remaining variables given its neighborhood. The neighborhood size is penalized by the parameter λ, as increasing λ increases the sparsity of the coefficients. The parameter λ is chosen empirically to produce almost the same degree of sparsity given different equipments. Figures 5.4a, 5.4b, and 5.4c shows the resulting graphs. The selected λ and the generated graphs were computed following the procedure described in [65].

The relationships found in the graphs from Figure 5.4 corroborate with the previous hypothesis that the differences on behavior found on different states could be used to assess the current state. Also, while there are relationships that persist between the distinct states, the number and the neighborhoods found were quite different between them. As an example, while the pair of variables $V19$ and $V20$ produces a neighborhood during the normal state, the same cannot be said during the anomalous state, where $V19$ is linearly independent from any variable and $V20$ is included on the largest neighborhood, which is a different behavior. This difference is also found in the shutdown state, where $V19$ and $V20$ are still in the same neighborhood, but not the same neighborhood found during anomalous states. These and other relationships are not clearly visualized when examining the correlation matrices in Figure 5.3.

Lastly, a hierarchical clustering algorithm was applied to the correlation matrices. *Clustering* is an unsupervised learning problem where the samples, or attributes, are grouped based on their similarity [66]. Since correlation measures linear dependency between the attributes, correlation can be considered a similarity metric and can be used to group the attributes.

However, in order to use a clustering algorithm, a dissimilarity measure is necessary. As such, the correlation matrices were converted to Pearson correlation



(a) Normal state partial correlation graph.

(b) Shutdown state partial correlation graph.

(c) Anomalous state partial correlation graph.

Figure 5.4: Partial correlation graphs for the three states: normal, shutdown, and anomalous.

distance matrices [67] as follow

$$\mathbf{C}_x\left(i,j\right) = \sqrt{2\left[1 - \mathbf{R}_x\left(i,j\right)\right]} \tag{5.9}$$

such that each entry $\mathbf{C}_x\left(i,j\right)$ of matrix $\mathbf{C}_x$ represents the dissimilarity between the pair of variables i and j. This matrix as then used as input for a hierarchical clustering algorithm.

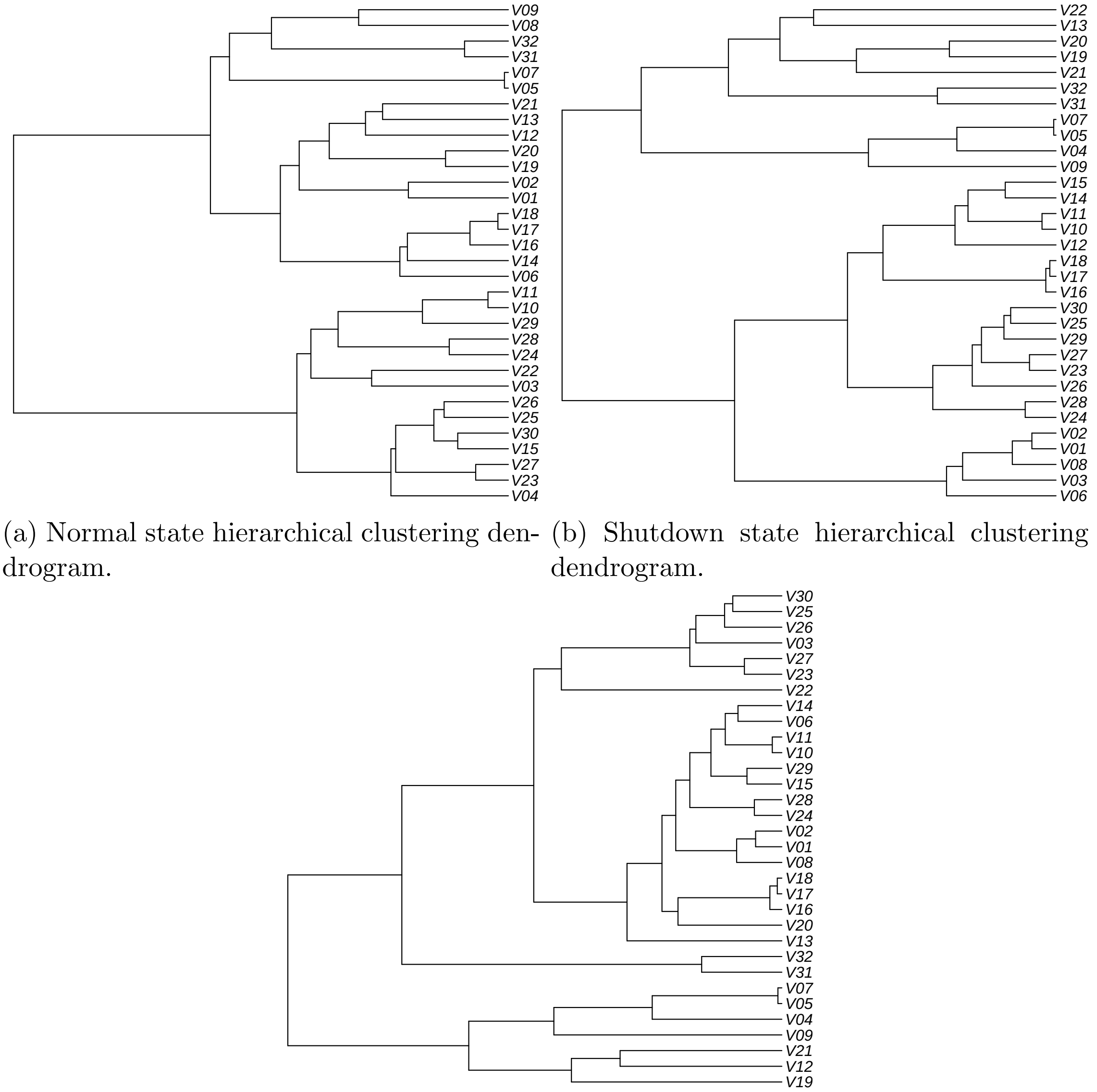
Hierarchical clustering is a process which aggregates data samples or attributes in multiple *clusters* (groups) in a structured fashion, which can be represented by a tree-like structure named *dendrogram*. This representation allows to observe how the different elements are organized at different levels. The pseudo-code of the employed algorithm is described in Algorithm 5.1.

Algorithm 5.1 Hierarchical agglomerative clustering algorithm pseudo-code.

```
1. Starting with each element as an independent cluster $\mathcal{C}_k = \{\mathbf{x}_k\}$. In the first
   iteration, $C_0 = K$, where $K$ is the number of elements and $C_0$ is the number
   of clusters;

2. Compute the dissimilarity between each cluster. This computation can be
   made using many methods [66, 68]. During this work the Ward's method was
   used [69];

3. At iteration $n$, find the most similar pair of clusters $(\mathcal{C}_i, \mathcal{C}_j)$. This pair will
   be aggregated in a single cluster $\mathcal{C}_k = \mathcal{C}_i \bigcup \mathcal{C}_j$, decrementing the number of
   clusters $(C_n = C_{n-1} - 1)$;

4. Repeat steps 2 and 3 until there is only one cluster containing all $K$ elements.
```

The resulting dendrograms obtained for the normal, shutdown, and anomalous states are presented in Figures 5.5a, 5.5b, and 5.5c, respectively. Comparing the structure found on the dendrograms and the graphs found on Figure 5.4, we observe that the groups that occurred in the partial correlation analysis also occurred in the dendrogram, for the same states. Also, the changes in behavior between the different states also appears in the dendrogram. Lastly, while the differences between the two main groups are clear for the anomalous state (Figure 5.5c), they are weaker in the normal state (Figure 5.5a), producing almost uniforms clusters, while the shutdown groups behavior (Figure 5.5b) are in-between the normal and anomalous groups.

While the results of the exploratory analysis are only qualitative, these results clearly suggest the possibility of detecting changes in the equipment behavior by observing changes in the relationship between distinct attributes. This corroborates with the possibility of describing the underlying process only using data-driven models. In the next section we present the cross-validation procedure to assess and select



(a) Normal state hierarchical clustering dendrogram.

(b) Shutdown state hierarchical clustering dendrogram.

(c) Anomalous states hierarchical clustering dendrogram.

Figure 5.5: Hierarchical clustering dendrograms for the three states: normal, shutdown, and anomalous. The height of each line is proportional to the dissimilarity between daughters nodes.

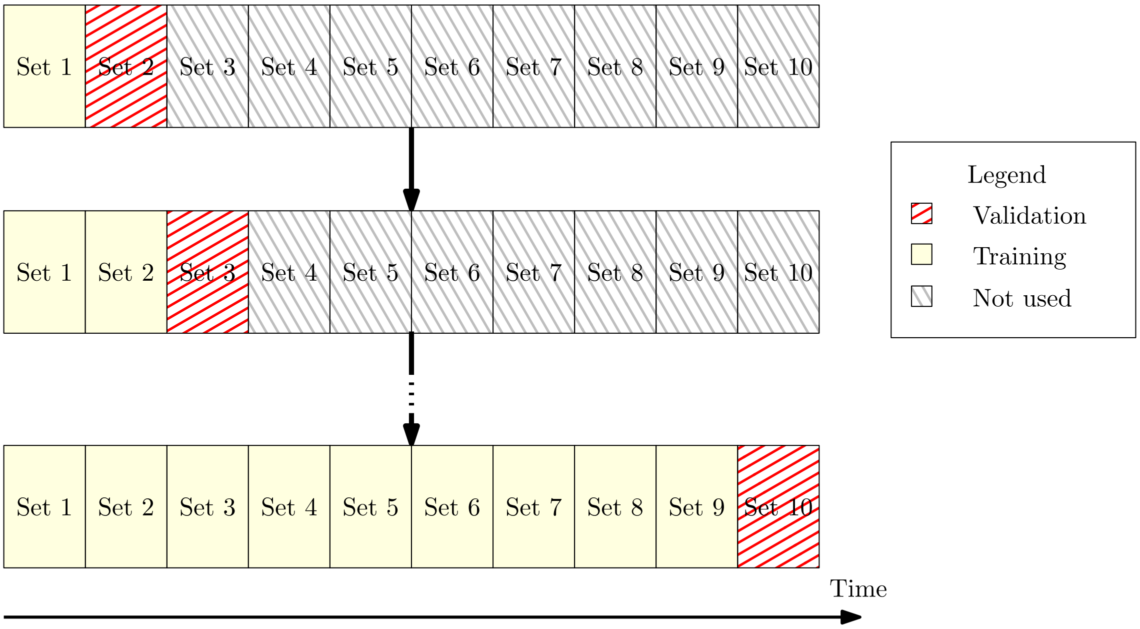


Figure 5.6: Temporal K-fold with $K = 9$.

the best models employed to achieve this aim.

5.3.3 Cross-validation procedure

This section presents the cross-validation procedure used to evaluated the proposed system. As previously discussed, first we divided the dataset in two disjoint sets: the *test set*, consisting of data from equipment 3, and the *training set*, consisting of the remaining equipment data.

To select the best set of parameters for the SBM model, we followed a grid search procedure using K-fold cross-validation to evaluate each model. However, given the temporal nature of the dataset, the normal K-fold method could produce biased results, as temporal series can have trends, cyclical, seasonal components, or other relationship between neighbor samples [70]. A traditional K-fold approach assumes independently and identically distributed (i.i.d.) samples to produce a reliable estimate of the model performance. Temporal series can violate the independence assumption, producing overoptimistic estimates of the model performance [71]. To cope with this problem, we followed a modified K-fold approach: considering that time-series samples are indexed in increasing time order, we divide the series in $K + 1$ time ordered groups, were the kth group is succeed by its futures values. Then, the k model is trained with the previous k partition and validated with the next $k+1$ partitions in a forward chaining manner [71]. This procedure is illustrated in Figure 5.6 for $K = 9$.

Given the extension of the database, this procedure was used with $K = 3$, with the training set decimated by 30. As each equipment from the training set has a total of 657.961 samples, this decimation is necessary to make the training procedure feasible, and did not affect the performance of the models. Also, considering that we have multiple equipment in the training set, samples from the same time-interval were taken from each fold and concatenated to produce the training and test sets. The best model obtained from the grid search procedure was then evaluated in the test set, producing the system final performance.

5.4 Results and discussion

This section presents the results obtained during the validation and test procedures. To select the best model to the proposed task we considered all the following system variations:

- We used only the full SBM formulation (Eq. (3.21)) as AAKR models achieved similar or inferior performance during preliminary tests;
- Choice of similarity function, as presented in Section 3.2, with distinct values of $\gamma \in \{0.01, 0.1, 1\}$ and multiple norms: ℓ_p norm with $p \in \{1, 2\}$, and the Mahalanobis distance;
- Classification solely based on the SBM model;
- SBM strategy for selecting the prototypes: standard method [43] with decimation factor $s \in \{1440, 2160, 2880\}$; the proposed threshold-method for threshold values $\tau \in \{0.05, 0.25, 0.495\}$; and the booststraping prototype selection method (see Section 3.4) with $\tau \in \{0.05, 0.25, 0.495\}$ and $s \in \{1440, 2160, 2880\}$;

While the full combination of the above parameters leads to a large number of possible configurations, in this section we present results using all combinations. The method based in [4] and presented in Section 3.4 was not employed in this chapter since it requires the distance matrix to compute the best set of prototypes. Given the number of samples in the training set, the necessary memory to produce this matrix would make this method infeasible. As such, we used the proposed booststraping version of the model also presented in Section 3.4.

5.4.1 Cross-validation results

This section presents the results obtained using the cross-validation procedure. Before we present the best model or set of models, first we discuss the influence of the parameters on the model performance and complexity. Six parameters were studied: the prototype selection method; the chosen similarity function; the distance metric employed; the kernel width parameter γ; and two parameters related with specific prototype selection methods, the threshold factor τ and the decimation factor s.

Prototype selection methods

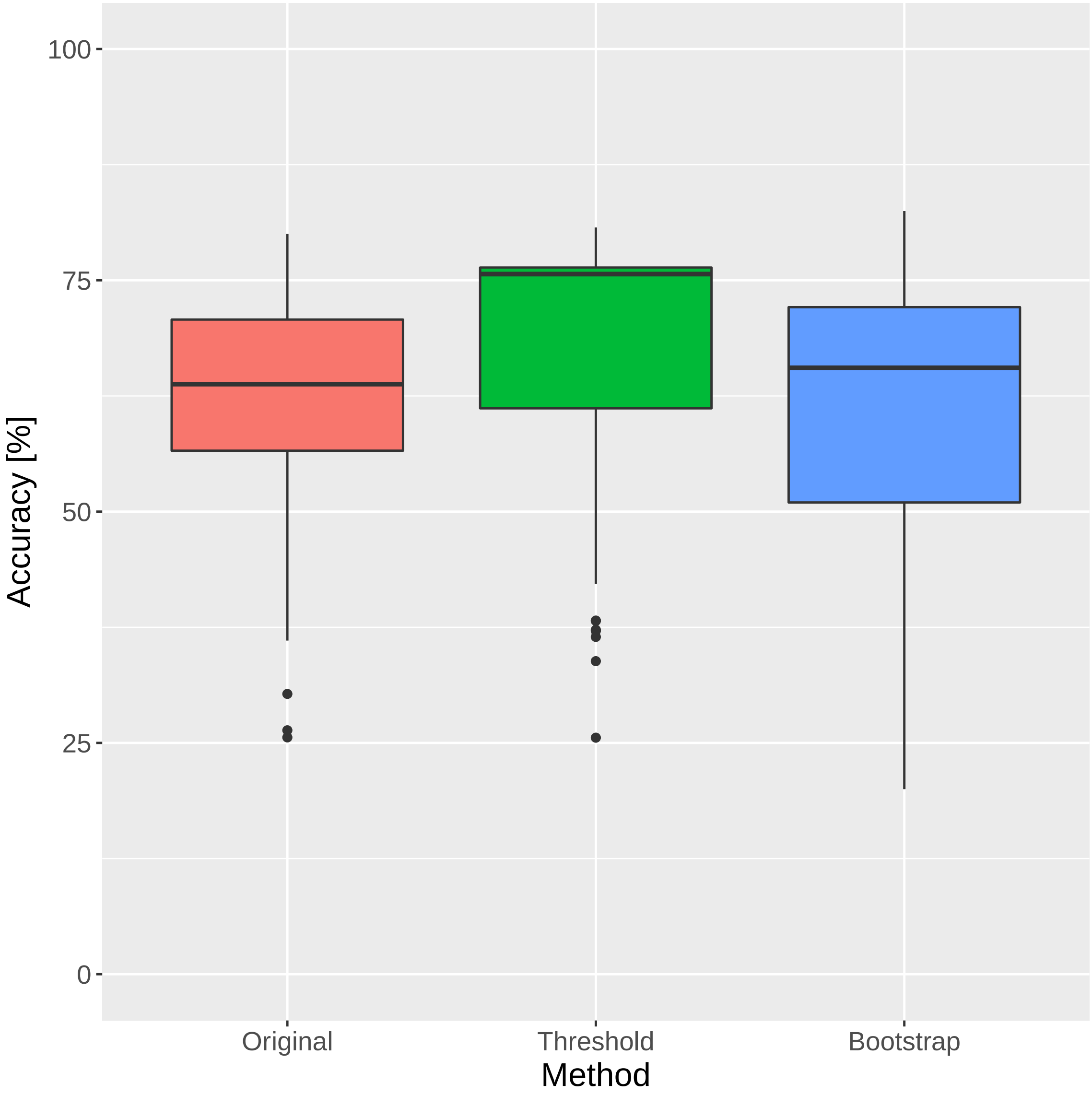


Figure 5.7: Accuracy box plot for the employed prototype selection methods.

Figure 5.7 presents the accuracy box plots for the employed prototype selection methods. A *box plot* is a standardized way of displaying the distribution of a sample, indicating its degree of dispersion and skewness, as well as possible outliers [63]. The box bottom and top are the first and third quartiles, and the band inside the box is the median. The upper whisker, in this case, extends from the third quartile to the highest value within 1.5 of the inter-quartile range (IQR), while the lower whisker extends from the first quartile to the lowest value within 1.5 of the IQR. The *inter-quartile range* is the distance between the first and third quartiles, and measures the statistic dispersion. Values outside this range are considered outliers and plotted as small dots [63]. Each factor is represented by a different box. In Figure 5.7, each prototype selection method is a factor.

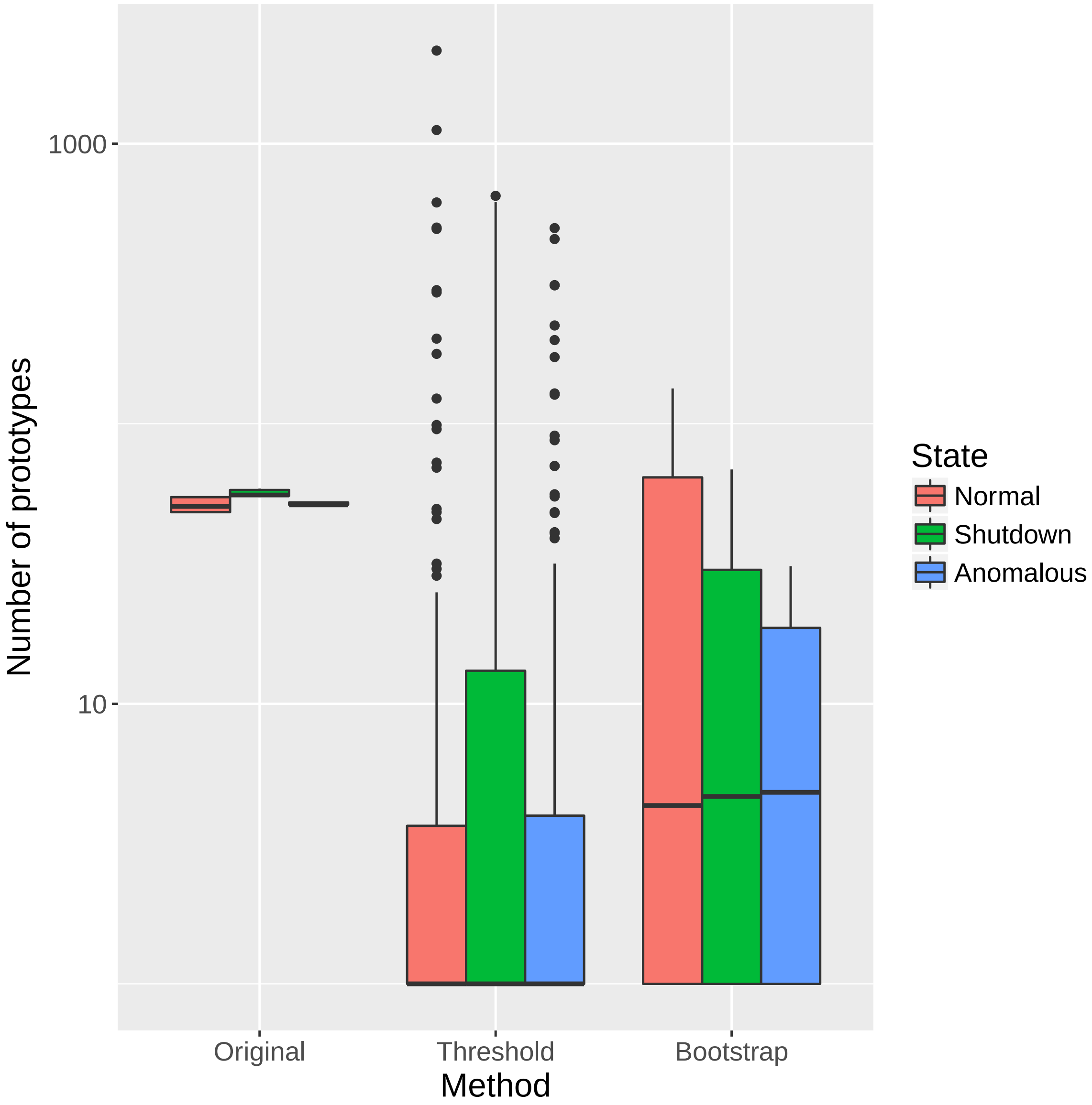


Figure 5.8: Box plot of the number of prototypes for the employed prototype selection methods. The number of prototypes axis is in logarithmic scale.

As illustrated in Figure 5.7, the different methods are almost equivalent in classification accuracy, with the threshold method achieving the best results in a more consistent manner, followed by the original method and by the bootstrap method, less consistent but producing the best model given its upper whisker. While the difference in accuracy is small, the difference in complexity is clear, as shown in Figure 5.8, with the original method achieving the highest complexity in number of prototypes. This result is due to the selection procedure, which is proportional to the number of samples in the training set. The median number of prototypes for each class in the original method is almost 10 times greater than the one of the bootstrap method and almost 100 greater higher than the one of the threshold method. These results demonstrate the efficacy of the proposed methods in reducing the computational complexity without losing accuracy.

Still, the results presented in Figure 5.8 also allow some important observations. While the original method produces more prototypes, the number of prototypes is almost deterministic, with a very small dispersion, mostly likely caused by changes in the decimation parameter s. As for the other two methods, they can produce a greater variability in number of prototypes and, in some cases, achieving values higher than ones of the original method, where the threshold method produces a number of prototypes 10 times greater than the ones of the original method. Although these occurrences are uncommon, they should be treated with caution, as an excessive number of prototypes could impair the system performance, increasing its computational complexity.

Lastly, looking at the dispersion considering each class, while there is some difference in the number of prototypes for the threshold method, their distribution shows little variation. This indicates that, while the classes can be clearly separated, the difficulty in describing each class is almost the same.

Similarity functions

Figure 5.9 shows the accuracy box plot for each similarity function. The original SBM similarity function described in [36] was the one that achieved the best accuracy in the most consistent manner. However, there is an overlap between most of the similarity functions. These results imply that, while it should exist an optimal similarity function for this application, the correct choice is not critical, as the model is robust against sub-optimal decisions.

Distance metric

Figure 5.10 presents the accuracy box plots for each distance metric. While there is no statistical difference in performance, these results demonstrate that the ℓ_2 norm was the most consistent, followed by the Mahalanobis distance. These results are very similar for different metrics. As such, for this application, the choice the optimal norm parameter is not as critical as one of the other parameters, such as the training method. In such case, the decision then falls into selecting the least computationally intensive metric. Considering this restriction, an ℓ_p norm would be the best option, as computing these distances does require computing cross-attribute terms.

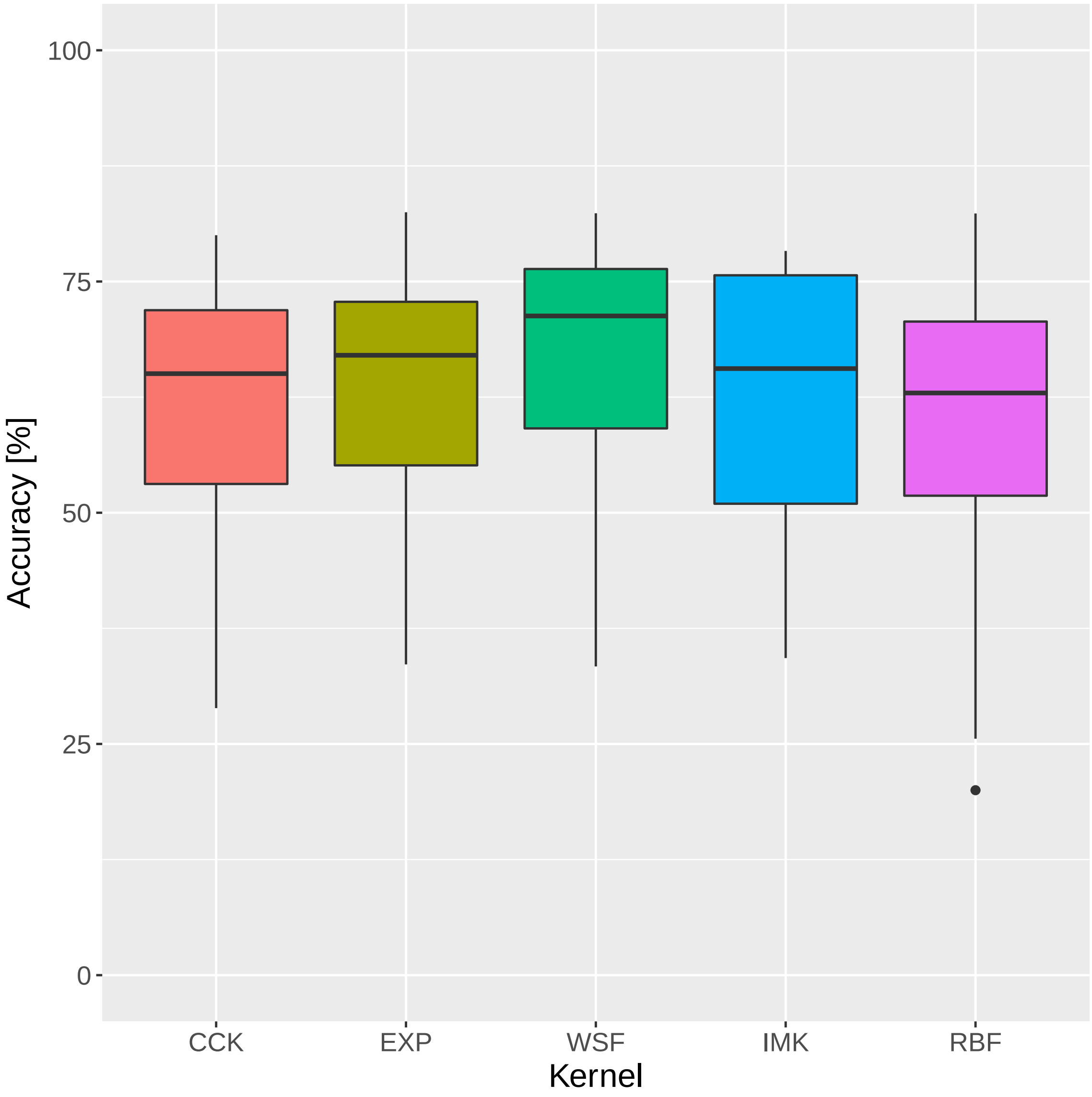


Figure 5.9: Accuracy box plot for the employed similarity functions.

Kernel width γ

The influence of the kernel width γ in the model accuracy is shown in Figure 5.11. As with the distance metric, these results are inconclusive. While there is a small increase in the accuracy for greater γ values, as the value increases, it reduces the radius where the similarity values are significant, increasing the number of prototypes necessary to describe the dataset. As $\gamma \rightarrow \infty$, each sample only describes itself, achieving a model with maximum accuracy in the training set, but without generalization power and with the largest complexity, producing overfitting. The inverse is also true. As γ tends toward zero, the radius of similarity increases, reducing the necessary number of prototypes. In the limit, only one sample would be selected for each class as prototype, most likely, the geometric median vector of each class, the point which minimizes the sum of distance to all other points.

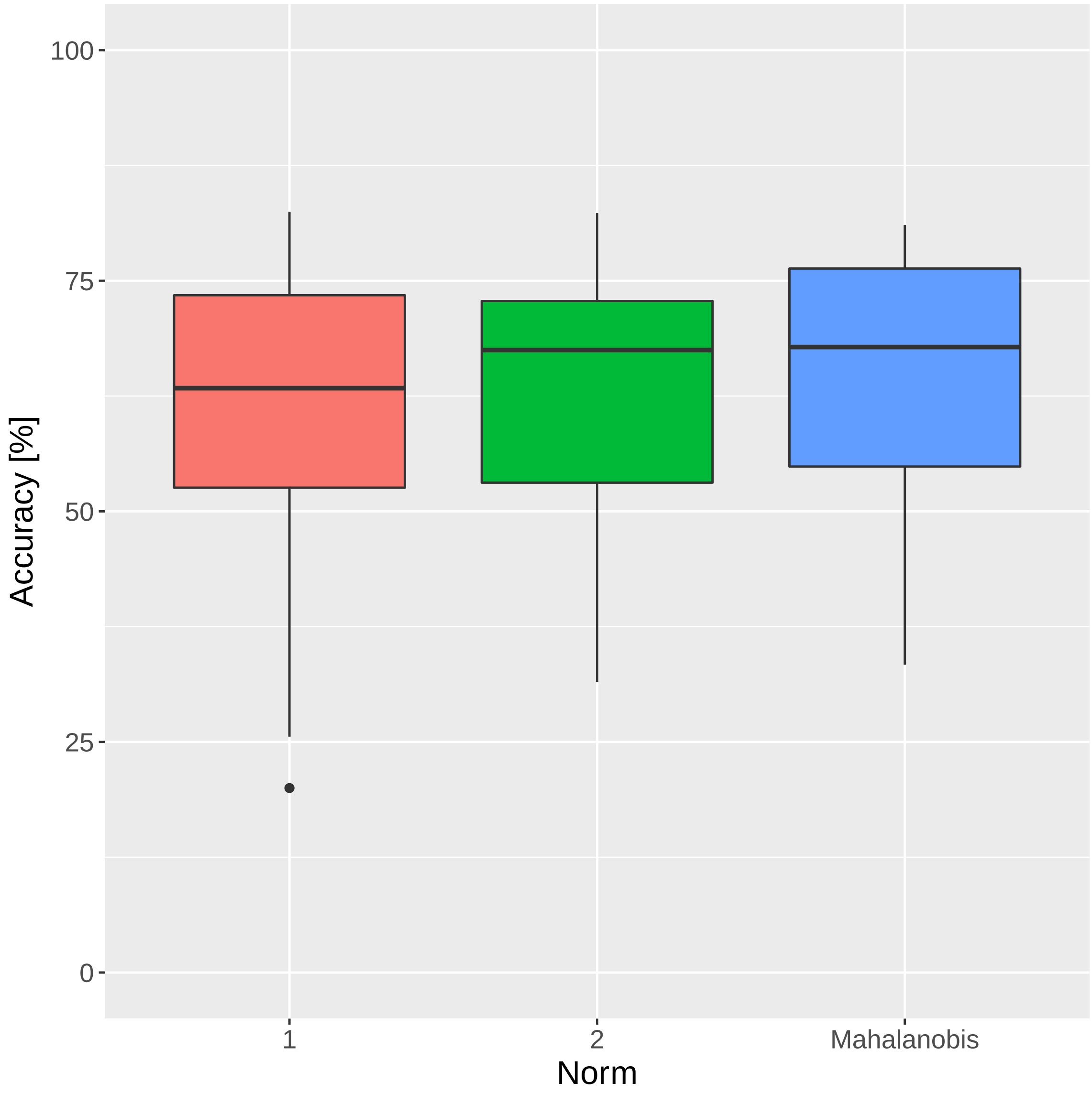


Figure 5.10: Accuracy box plot for the employed distance norms.

Threshold factor τ

The threshold factor parameter τ is shared by the threshold and the bootstrap methods. Figure 5.12 presents the accuracy box plot for this parameter for the threshold method. While this parameter does not produce great variation in the model performance over the chosen range, as with the γ parameter, when $\tau \to 1$ all the samples are selected as prototypes, reducing generalization power of the model. For $\tau \to 0$, only a small set of prototypes (often just one) would be selected, producing a sparse representation of the training set. This behavior is very similar to the effect produced by the γ parameter, and these parameters can be used together to mitigate the influence one another during training.

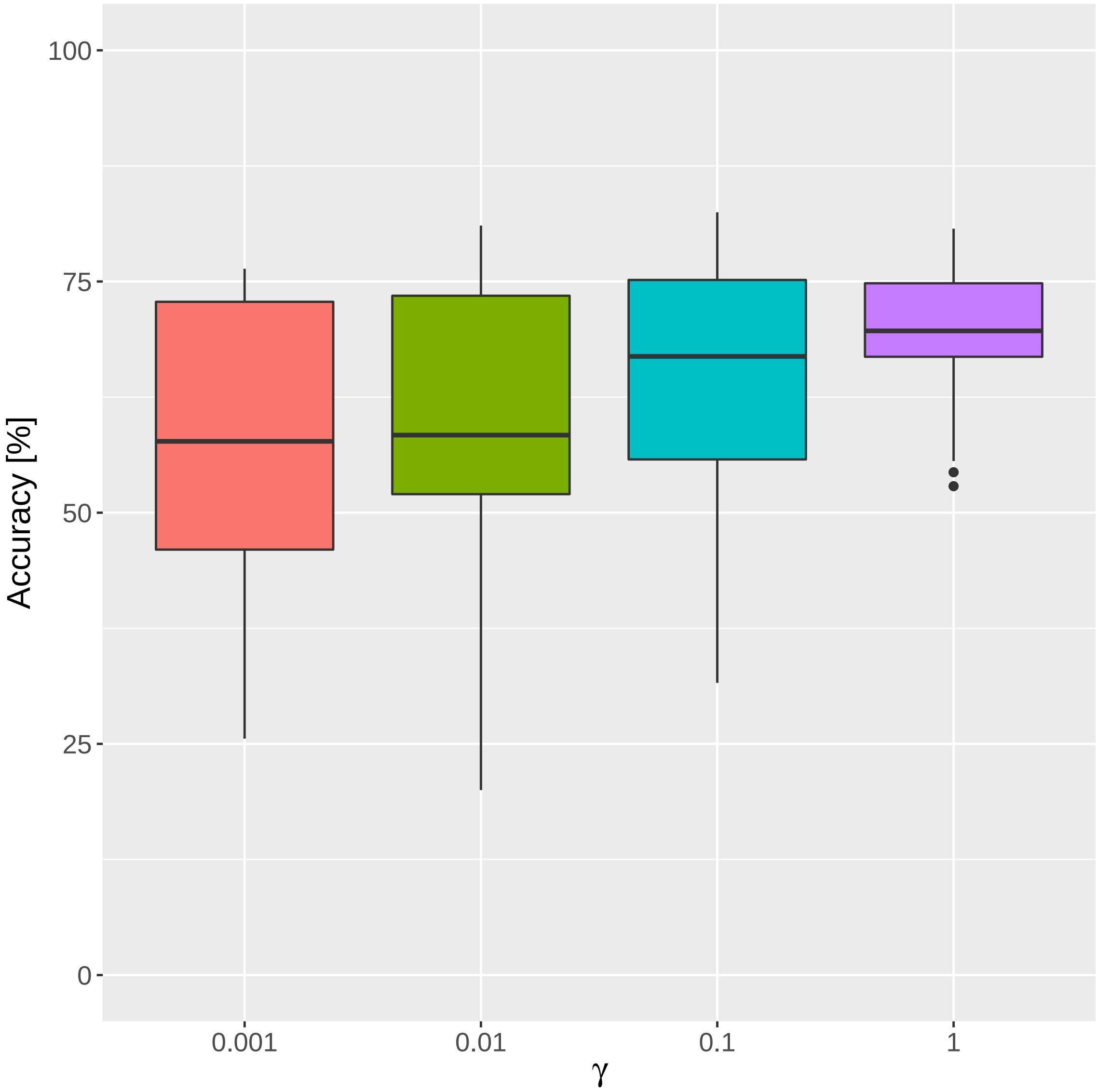


Figure 5.11: Accuracy box plot for different kernel width γ values.

Decimation factor s

The decimation factor s is shared by the original and bootstrap methods. As shown in Figure 5.13, the decimation factor produces inconclusive results, as there a subtle increase followed by a small decreasing in accuracy while the decimation values increases. A possible explanation for this behavior is that it is reasonable to consider as time-series samples in the same neighborhood are very similar. The decimation factor would then discard samples that represent the same phenomenon and do not add any new information. The remaining set is less similar and more informative, producing a more discriminative set of prototypes. However, as the decimation factor keeps increasing and reducing the number of samples, there would be an optimal point where accuracy would stop increasing and start to fall, as the remaining samples would stop describing the underlying distribution.

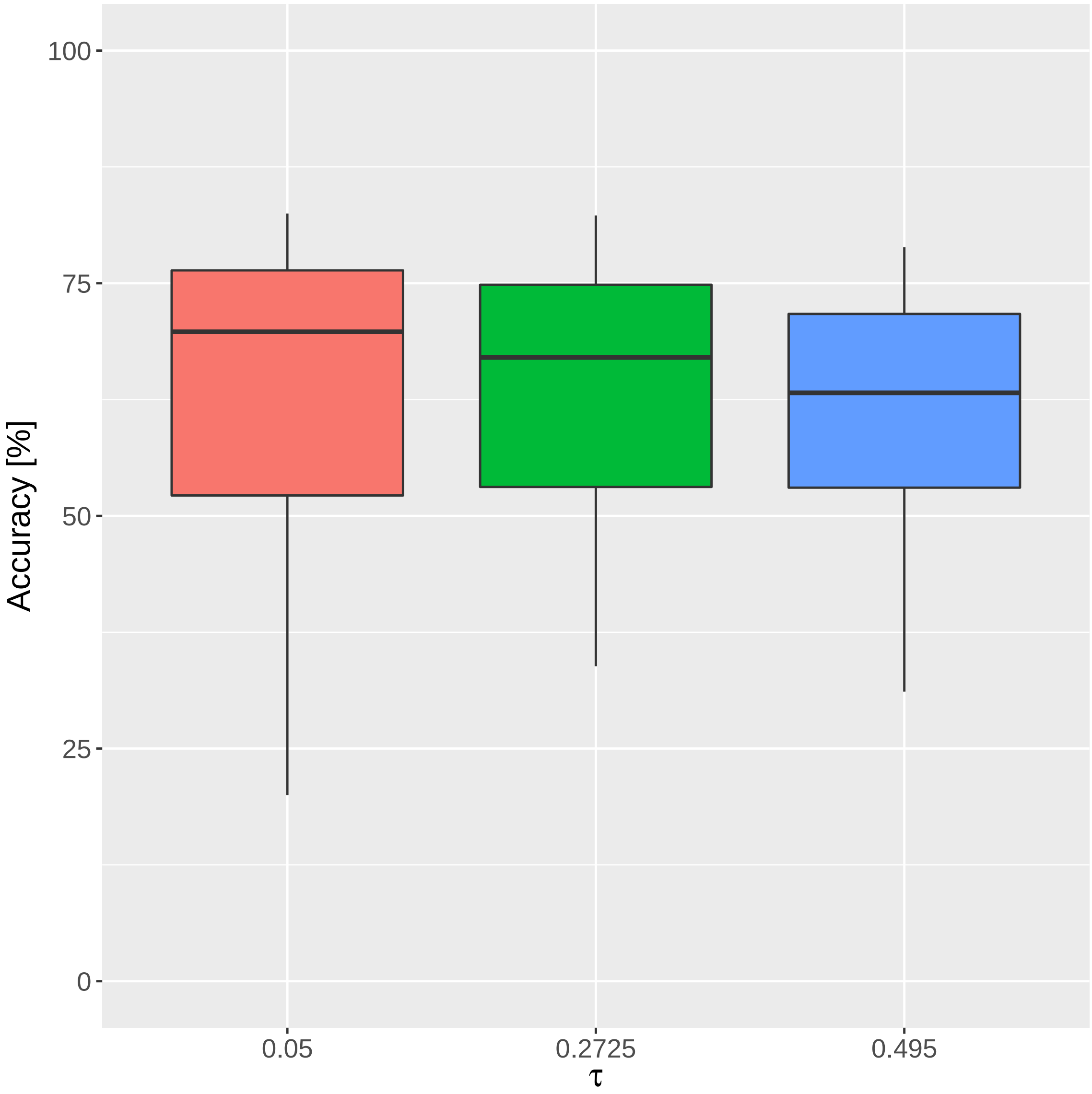


Figure 5.12: Accuracy box plot for interval of τ.

Model selection

This section presents the selected model given the results of the cross-validation experiments. Table 5.1 presents the 10 best models in descending order considering the obtained accuracy. The similarity functions used are WSF (Eq. (3.10)), RBF (Eq. (3.12)), IMK (Eq. (3.8)), CCK (Eq. (3.9)), and EXP (Eq. (3.11)). Surprisingly, the 12 best obtained results were from bootstrap method.

While the analysis of the validation results presented in Table 5.1 made some decisions clear, such as the prototype selection method as the booststrap method and the kernel width $\gamma = 0.1$, as the 10 best models are almost statistically equivalent, decisions pertaining to the remaining parameters are still indefinite.

The complexity of a given SBM is given by its number of prototypes. Therefore, to help defining the remaining parameters, we evaluated the complexity of the

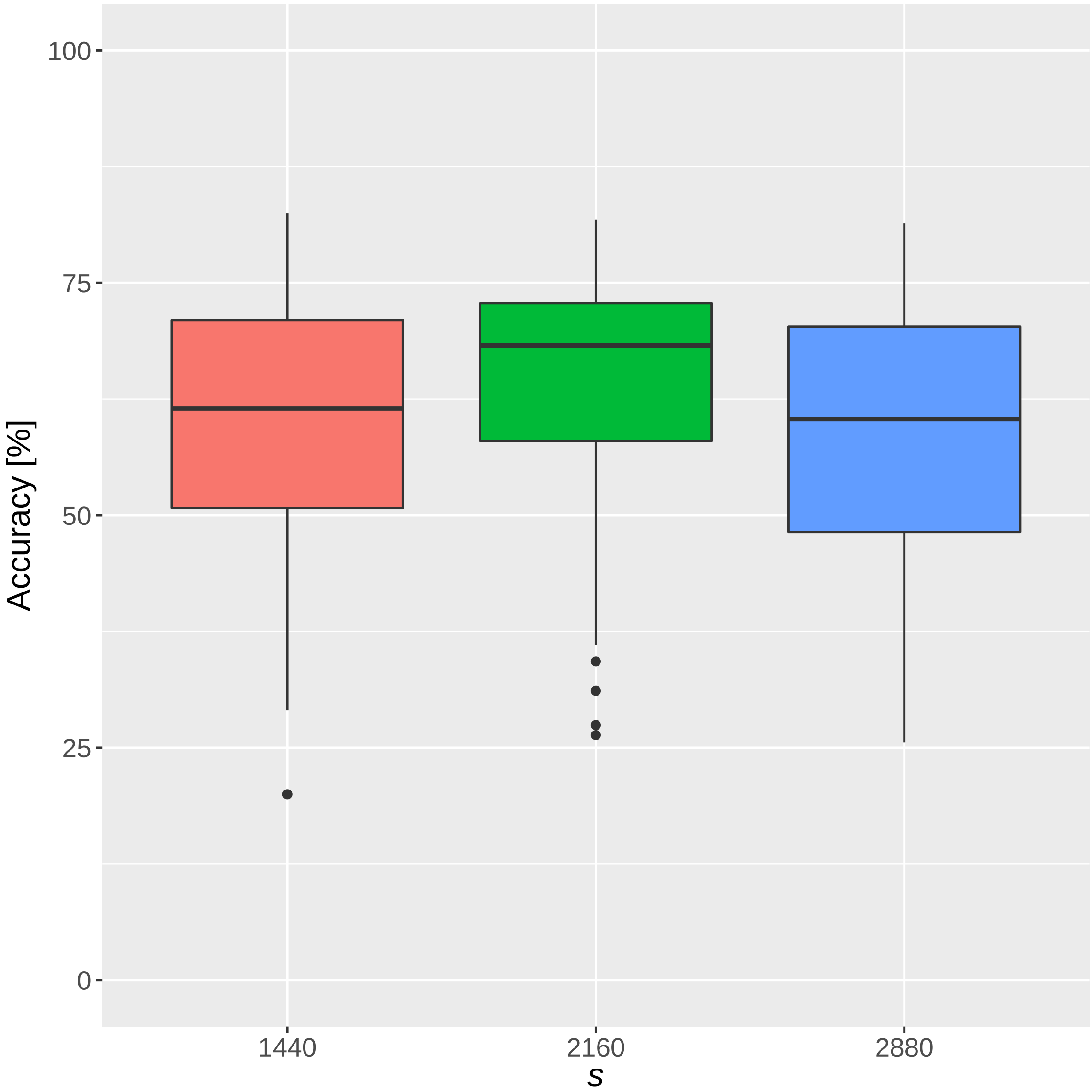


Figure 5.13: Accuracy box plot for the decimation factor (s).

models during the cross-validation procedure. Table 5.2 presents the number of prototypes for the 10 best models ordered in descending order considering the obtained accuracy. The number of prototypes is given for each class.

As shown in Table 5.2, while the models achieved almost the same accuracy, the number of prototypes employed for each model to describe the target classes changed a great deal for some classes. These results can be explained by changes on the similarity function, distance metric, or the similarity threshold τ. The most influential factor is the similarity threshold τ, as increasing τ is equivalent to increasing the radius ϵ in the bootstrap method. This produces prototypes covering a greater number of samples, demanding less prototypes without affecting the model accuracy. Also, considering the number of prototypes per class, one can observe that the normal state requires a larger number of samples, implying that this class is harder to describe. This can be due distinct operation modes being represented

Table 5.1: Cross-validation accuracy (%) for the 10 best SBM configurations.

Prototype selection method	Similarity Function	Distance Metric	γ	τ	t	Accuracy (%)
Bootstrap	EXP	ℓ_1	0.1	0.0500	1440	82.5 ± 3.6
Bootstrap	WSF	ℓ_1	0.1	0.0500	1440	82.4 ± 5.8
Bootstrap	RBF	ℓ_2	0.1	0.0500	1440	82.4 ± 3.8
Bootstrap	EXP	ℓ_1	0.1	0.2725	1440	82.3 ± 3.1
Bootstrap	EXP	ℓ_1	0.1	0.0500	2160	81.8 ± 2.4
Bootstrap	WSF	ℓ_1	0.1	0.0500	2160	81.8 ± 3.4
Bootstrap	EXP	ℓ_1	0.1	0.2725	2160	81.5 ± 2.2
Bootstrap	WSF	ℓ_1	0.1	0.0500	2880	81.4 ± 3.1
Bootstrap	EXP	ℓ_2	0.1	0.0500	1440	81.3 ± 2.9
Bootstrap	WSF	ℓ_1	0.1	0.2725	1440	81.3 ± 6.8

by the same label, with heterogeneous samples.

The analysis of the validation results presented in Tables 5.1 and 5.2 leads us to choose the configuration consisting of using the bootstrap method as prototype selection method, similarity function EXP, $\gamma = 0.1$, $\tau = 0.2725$, $t = 1440$, and ℓ_1 norm, as it produces the smallest set of prototypes with equivalent accuracy.

Table 5.2: Number of prototypes for the 10 best SBM configurations. Bootstrap prototype selection method with $\gamma = 0.1$. Nr – Normal state; Sd – Shutdown state; An – Anomalous state.

				Number of Prototypes		
Similarity Function	Distance Metric	τ	t	Nr	Sd	An
EXP	ℓ_1	0.0500	1440	125 ± 54	67 ± 25	24 ± 8
WSF	ℓ_1	0.0500	1440	125 ± 54	67 ± 25	24 ± 8
RBF	ℓ_2	0.0500	1440	64 ± 34	49 ± 16	18 ± 7
EXP	ℓ_1	0.2725	1440	43 ± 23	34 ± 8	15 ± 6
EXP	ℓ_1	0.0500	2160	87 ± 37	44 ± 14	19 ± 4
WSF	ℓ_1	0.0500	2160	87 ± 37	44 ± 14	19 ± 4
EXP	ℓ_1	0.2725	2160	37 ± 18	28 ± 8	12 ± 3
WSF	ℓ_1	0.0500	2880	67 ± 30	31 ± 12	15 ± 3
EXP	ℓ_2	0.0500	1440	88 ± 46	57 ± 21	18 ± 5
WSF	ℓ_1	0.2725	1440	34 ± 18	28 ± 8	15 ± 6

5.4.2 Results on the testing set

In this section we analyze the performance of the proposed system on the test set. For this study the model considered in the previous section was chosen, consisting of an SBM model using the bootstrap method as prototype selection method, similarity function EXP, $\gamma = 0.1$, $\tau = 0.2725$, $t = 1440$, and ℓ_1 norm.

As mentioned in Section 5.3, the test set consists of the remaining equipment, equipment number 3, which contains 25% of the samples total. Using the test set the proposed system achieved an accuracy of 91.38%. The confusion matrix is shown in Table 5.3.

Table 5.3: Confusion matrix for the test dataset. Nr – Normal state; Sd – Shutdown state; An – Anomalous state.

Class	**Nr**	**Sd**	**An**
Nr	535100	58	0
Sd	5193	66125	6262
An	30844	14372	7

These results correspond to what was expected. While the normal state is correctly described, achieving a precision of almost 100%, misclassifying only a small set of samples, the performance over the anomalous states had very poor accuracy, below 1%. Considering that the set of anomalous states is heterogeneous, with multiple different states under the same label, while the normal and shutdown events are mostly homogeneous, its reasonable that anomalous states are harder to classify. This characteristic is more critical when we consider that the test set consist of another equipment and, while the normal and shutdown and test states are shared between different pieces of equipment, anomalous states could appear in distinct formats in different pieces of equipment.

Also, the proposed models do not use any temporal information to produce a decision. As such, transitions between states are not treated as such, producing ambiguous labels which would reduce the system performance. This effect can be visualized in Figure 5.14, where the time series with unnormalized similarity scores are presented. One can see that the proposed SBM model produces very noisy similarity series. Besides, there is a decreasing trend in the similarity scores as time progress. This behavior can be a drift caused by some non-stationarities of the equipment, or it could be a hint of equipment degradation. In the former case, this drift could be mitigated by an adaptive model with memory, discarding older prototypes and adding new prototype samples during reliable periods.

Another possible cause of the low accuracy in detecting anomalous states can be the unbalanced nature of the used data set. Considering that training and test sets share the same class distribution, this means that anomalous samples are less than 7% of the training samples. Thus, this class could be underrepresented, requiring a higher number of samples to be correctly described.

However, these results indicates that the proposed system is capable of generalization, even achieving a test performance in a different equipment higher than the cross-validation performance (91.4% against 82.3%).

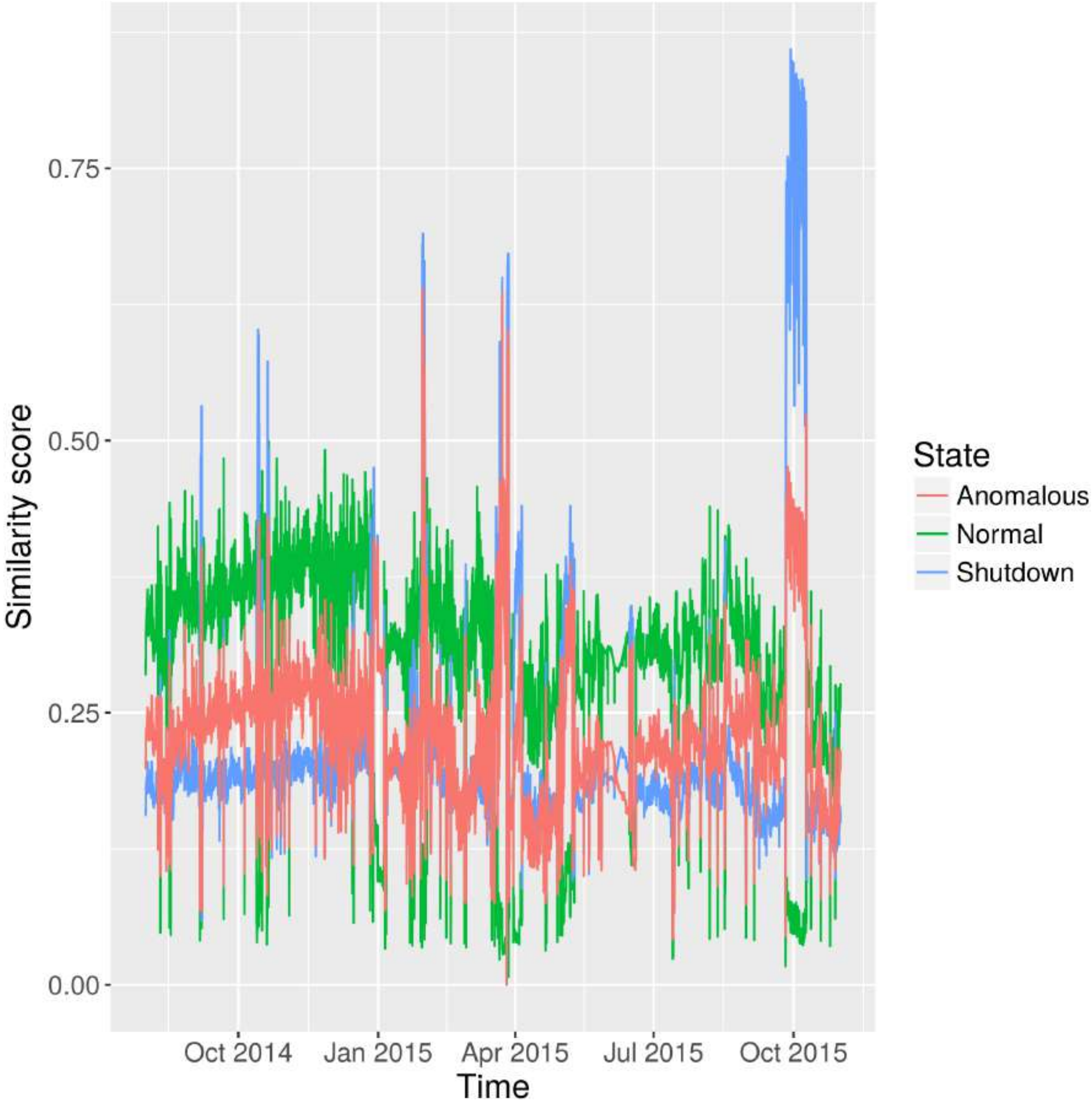


Figure 5.14: Time-series for the test set similarities scores.

5.5 Conclusion

This chapter addressed automatic fault diagnosis in oil platform equipment. The use of the similarity-based modeling was investigated as a diagnosis and monitoring model. The system was evaluated using cross-validation and tested using a procedure similar to cross-dataset validation. The employed database consist of multivariate time-series from four injection pumps of the same injection system on an oil platform, with samples taken minutely during the interval between August 01, 2014, and November 01, 2015, each pump producing 657961 samples, totaling 2631844 samples in the dataset. These multivariate series are composed of 41 attributes, from numerical to categorical attributes.

The proposed system was cross-validated using three pumps and assessed against the remaining pump, with the best model accuracy of 82.3% in the cross-validation procedure and 91.4% during the test procedure. While these results demonstrate the generalization power of the proposed method, there still room for improvement, as the model missed most of the anomalous events, indicating the need of (a) less ambiguous labels both during transitions and for the anomalous events and (b) the use of a temporal consistency model or temporal information to reduce the noise in the decision process and explore the temporal characteristics of the input series.

Chapter 6

Fault detection system

6.1 Introduction

This chapter presents an application based on the SBM framework: a fault detection system which receives signals from a given equipment and automatically checks its health. It learns prototypes, classes, and the similarity function with the operator's help. It is a complete application, composed of three different elements:

- a relational database, where all received and generated data is saved;
- a *data access and processing layer*, which receives the input measurements and computes their respective output;
- and a *data presentation layer*, a user graphical interface in a dashboard format presenting the current state of the equipment.

The proposed system permits monitoring and detecting novelties in real-time with the operator assistance. However, operator assistance is only needed when assessing detected faults or when introducing new prototypes or fault types. Thus, the proposed system frees operators to use their expertise to more challenging problems while also highlighting anomalous events or faults that could pass unnoticed otherwise.

To produce an informative system, the proposed system was devised considering that a user should be able to visualize prototypes, sensor importances, and sensor failures, and also edit or correct detected events or prototypes, in case of incorrect detections.

The proposed system is presented in the next sections. Section 6.2 details the user interface, including its usage. The data access layer, including the relational database and the learning algorithm, is described in Section 6.3. Lastly, Section 6.4 concludes this chapter.

6.2 User interface

As described before, the user interface must satisfy the following requirements: be informative, enabling an operator to visualize any anomalous or fault event; allow editing or deleting incorrectly classified or false positive detections; give contextual information, such as different signals importance and historical behavior; and, lastly, be user friendly. To met these requirements, we designed the user interface as a web interface divided in four windows:

1. **Similarity scores**: Presents the historical similarity score data for currently known event types, such as distinct faulty or healthy states;

2. **Events**: Presents detected event historical data, including the similarity score behavior during each event;

3. **Prototypes**: Gives information about the current set of prototypes and new prototypes to be added;

4. **Signals**: Exhibits signal importance and response.

Each window is described in their respective sessions as follows: Section 6.2.1 introduces the similarity score window; the event window and its functionalities are presented at Section 6.2.2; Section 6.2.3 briefly describes the prototype window and its functionalities; lastly, the signal window is discussed at Section 6.2.4.

6.2.1 Similarity score window

Depicted in Figure 6.1, it is the first window of the application. It is composed of two elements: a navigation menu, in the left, and a plot presenting the similarity scores over time for each class. The menu in the left allows a user to move between the different windows and appears in each window. It also presents a brief description of each window purpose.

The plot is the main element of this window. It presents historical data of the scores of each class, depicted in different colors, therefore allowing a user to understand when a fault occurred or when the equipment behavior changes.

Figure 6.1 also presents the special case when a new class is detected, labeled as *Unknown*. Following the procedure to be presented in Section 6.3.2, the first received sample is selected as a representative state of the *Default* condition. After that, new prototypes which are not identified as belonging to any class are classified as *Unknown*.

In these cases, the system produces an alarm and requires operator action to identify the new prototype as either a new class or as an existent class. However, the application continues to work while waiting for the operator action.

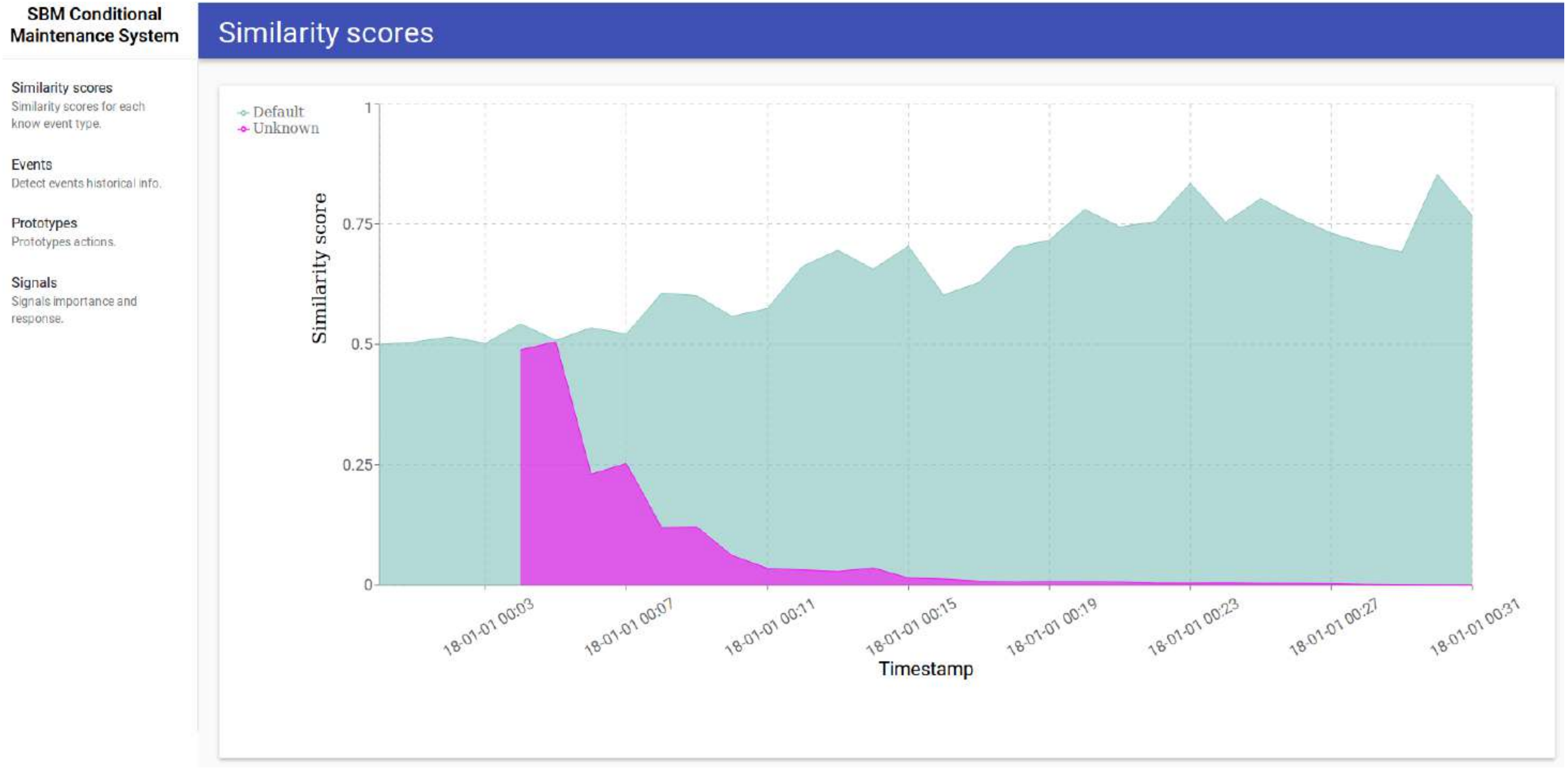
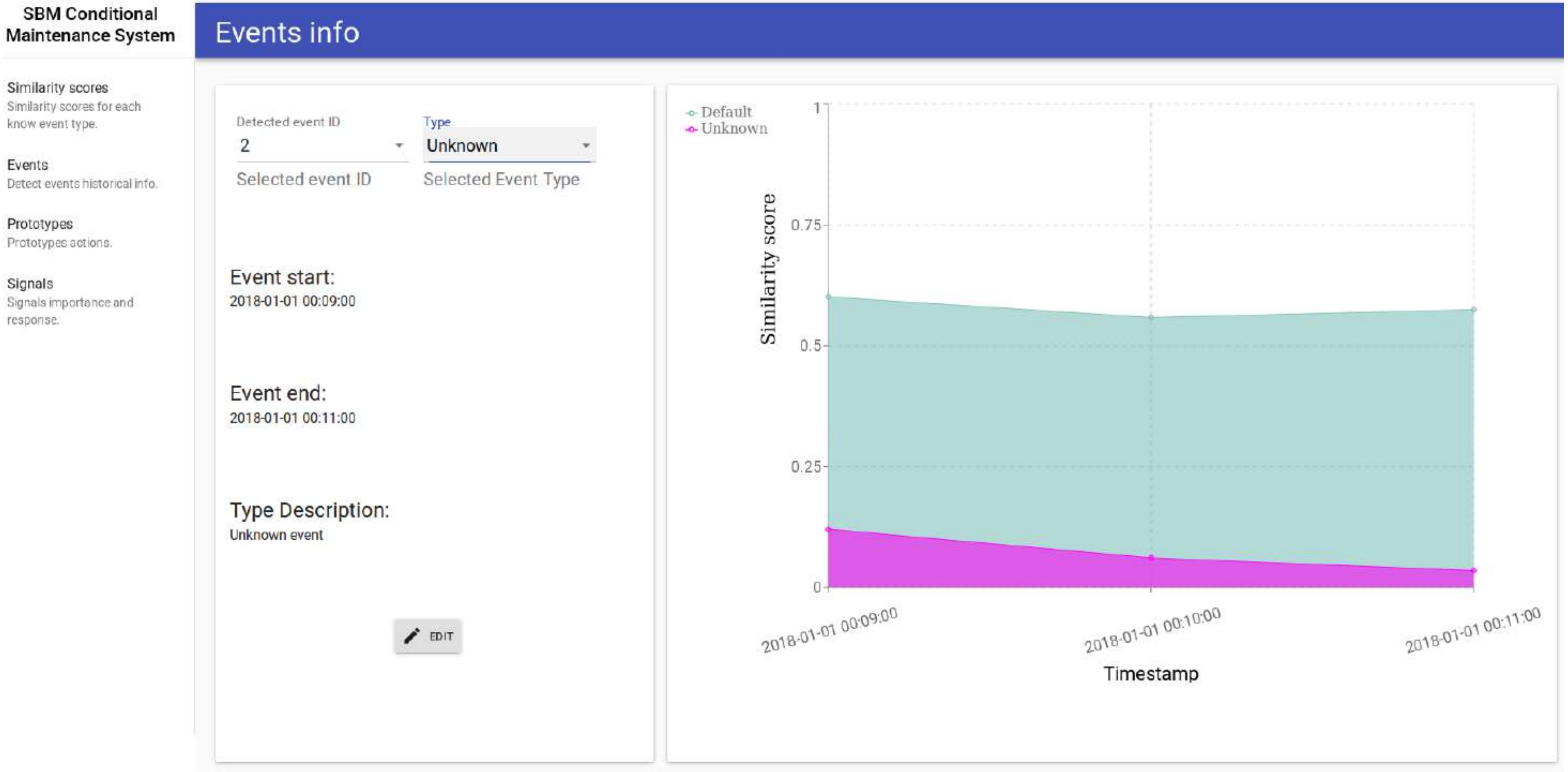


Figure 6.1: Similarity score window. It presents, at each instant, each known condition score given by the detection system.



SBM Conditional Maintenance System

Similarity scores
Similarity scores for each know event type.

Events
Detect events historical info.

Prototypes
Prototypes actions.

Signals
Signals importance and response.

Events info

Detected event ID: 2

Selected event ID

Type: Unknown

Selected Event Type

Event start:
2018-01-01 00:09:00

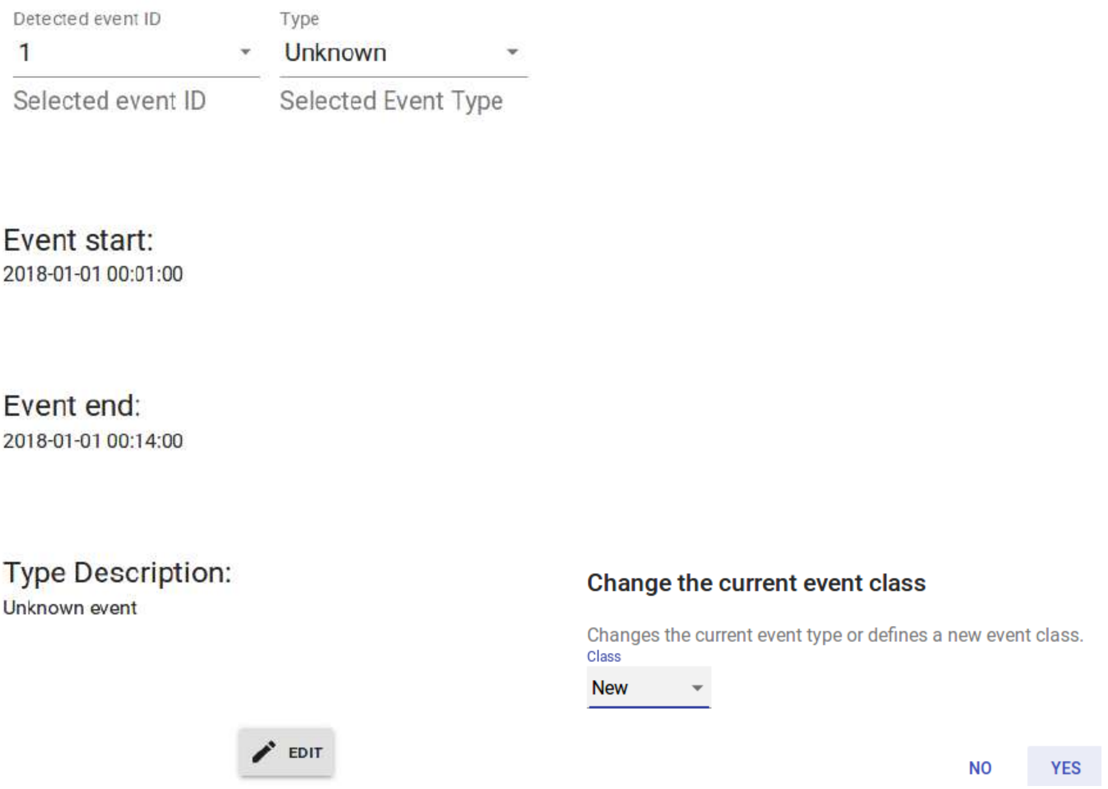
Event end:
2018-01-01 00:11:00

Type Description:
Unknown event

Figure 6.2: Event window. It allows a user to select a detected event, edit, and review its historical data.

6.2.2 Event window

The event window, shown in Figure 6.2, is composed of three elements: the common menu, in the left; the event info dialog, in the middle; and the selected event similarity score data, in the right.



Detected event ID
1
Selected event ID

Type
Unknown
Selected Event Type

Event start:
2018-01-01 00:01:00

Event end:
2018-01-01 00:14:00

Type Description:
Unknown event

(a) Events info dialog.

Change the current event class

Changes the current event type or defines a new event class.

Class
New

(b) Events type edition dialog.

Figure 6.3: Events' information and edition dialogs. It allows a user to review or edit a detected event.

The info dialog, detailed in Figure 6.3a, allows an operator to select a historical event, identified by a unique event id, and retrieve information about the selected event, such as: the detected event type, the timestamp where this event was first detected, when this event ended, and the event type description.

It also allows an operator to edit an event, as pressing the *edit* button opens the event edition dialog, shown in Figure 6.3b, where the operator can change the detected type to another predefined type, or define a new type of event. When the *New* option is selected, it opens another dialog where a user can input the new event and select its type. There are four possible types:

- *normal*: one of the healthy states;
- *shutdown*: all states where the equipment was turned off;
- *fault*: faulty states;

- *unknown*: not-yet-labeled detected anomalous states.

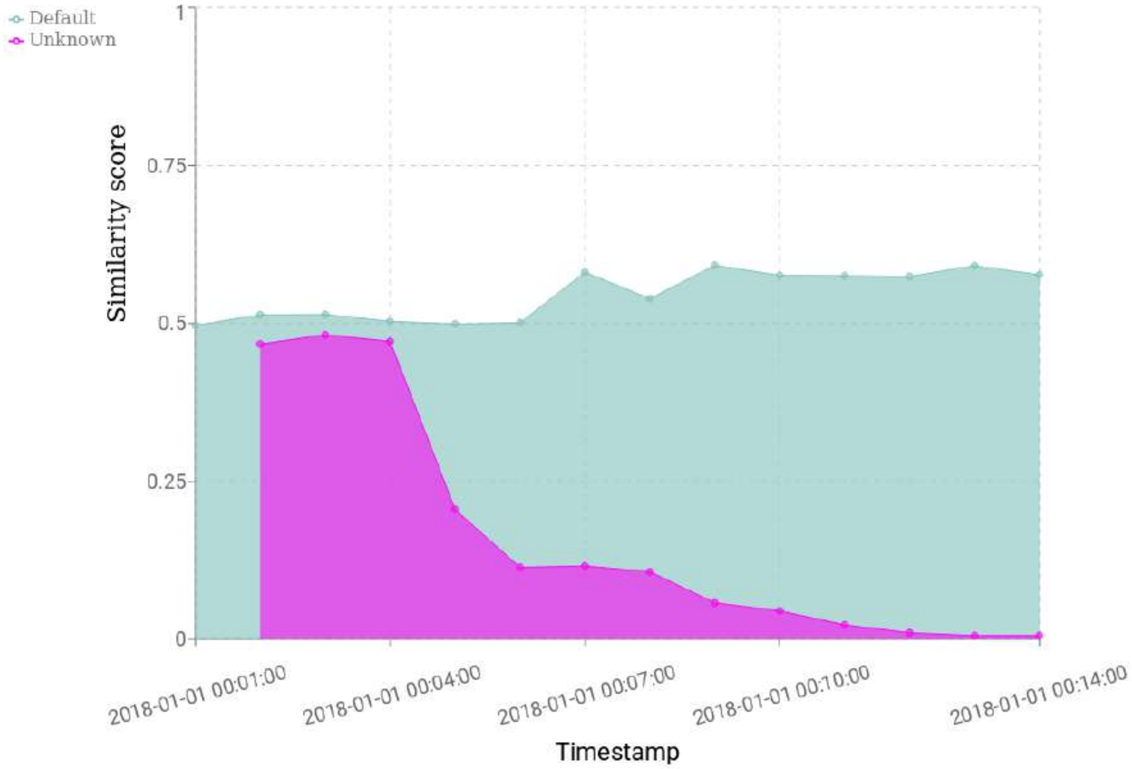


Figure 6.4: Similarity scores for a chosen event. This chart permits visualizing historical similarity score behavior during an event.

Lastly, Figure 6.4 presents the similarity scores during a chosen event, which permits a fast visualization of how the event occurred and developed.

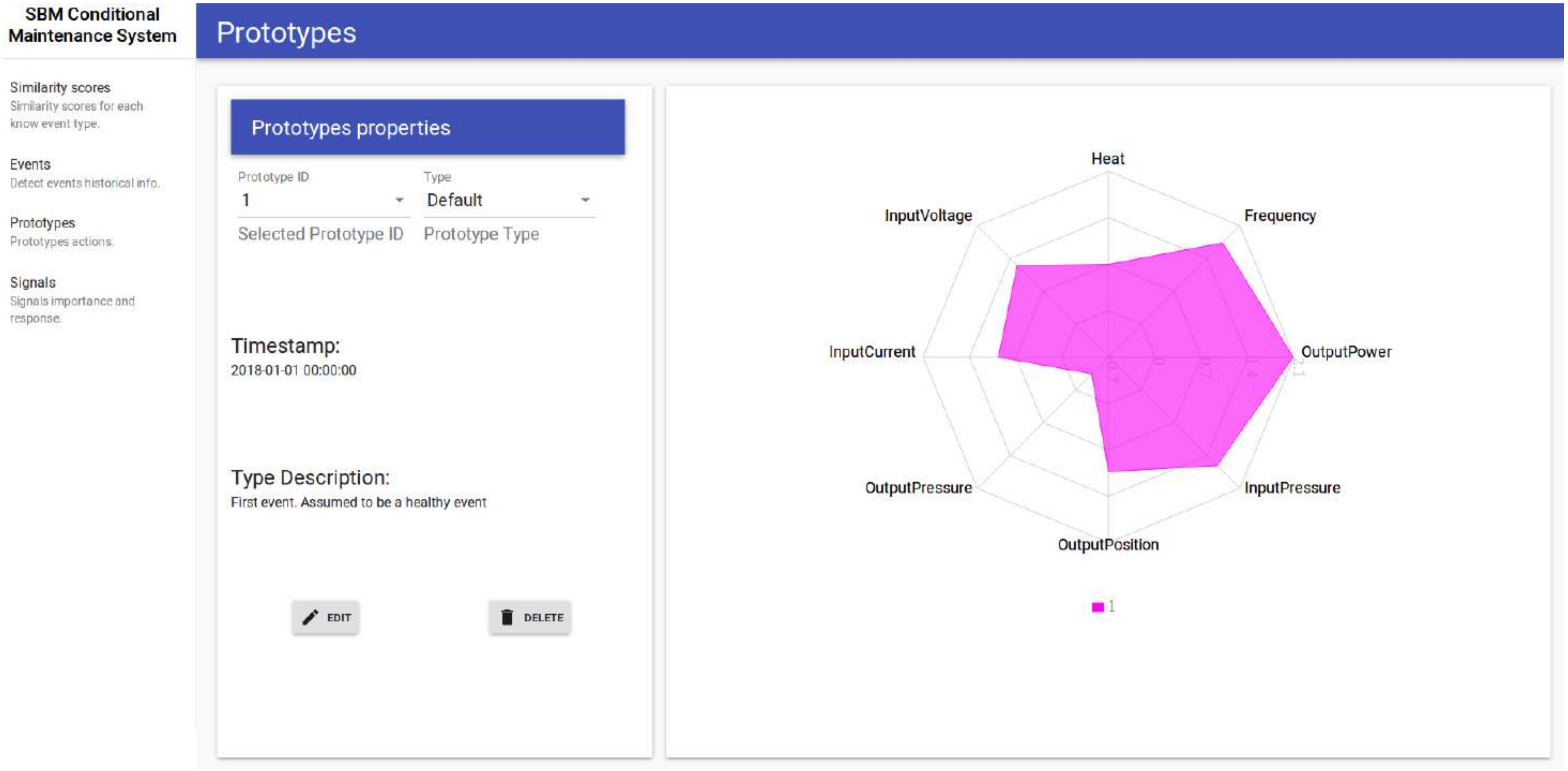
6.2.3 Prototype window

Figure 6.5 presents the prototype window, composed of three elements: the common menu, the prototype dialog, and the selected prototype radar chart [72].

The info dialog (Figure 6.6a) presents the characteristics of a given prototype, including when it was added, its identification, type, and type description. Also, it can be used to edit or delete the selected prototype, as shown in Figures 6.6b and 6.6c, respectively.

The edition procedure is very similar to the one described above for *events*. However, changing a prototype does not affect past decision as in the event cases, only future decisions. Also, deletion is irreversible, as deleting removes a prototype permanently. Like the edition procedure for prototypes, deleting a prototype does not affect historical data, only the system response after the deletion.

Lastly, the selected prototype radar chart, presented in Figure 6.7, helps an operator to visualize all features from a prototype in a single chart. This is useful for visual inspection and comparison between prototypes.



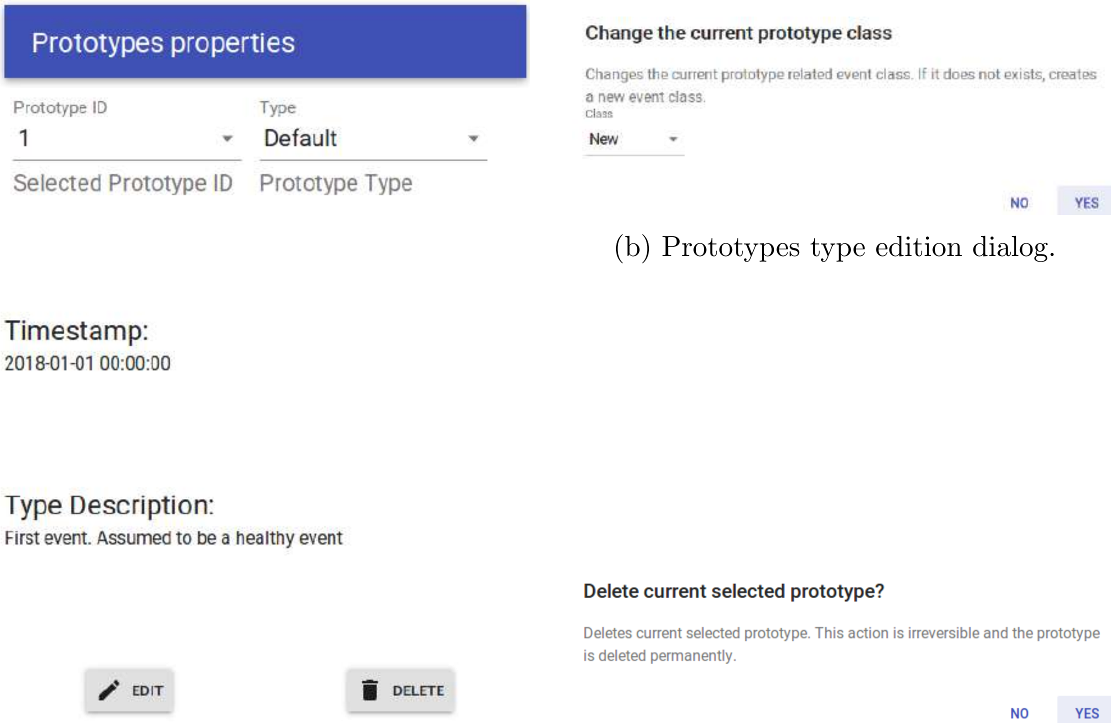
Prototype ID: 1 — Selected Prototype ID

Type: Default — Prototype Type

Timestamp:
2018-01-01 00:00:00

Type Description:
First event. Assumed to be a healthy event

Figure 6.5: Prototype window. It allows to select a prototype for visualization, displaying relevant information, and for edition or deletion.



Prototypes properties

Prototype ID: 1

Type: Default

Selected Prototype ID

Prototype Type

Timestamp:
2018-01-01 00:00:00

Type Description:
First event. Assumed to be a healthy event

Change the current prototype class

Changes the current prototype related event class. If it does not exists, creates a new event class.

Class: New

(b) Prototypes type edition dialog.

Delete current selected prototype?

Deletes current selected prototype. This action is irreversible and the prototype is deleted permanently.

(a) Prototypes description dialog.

(c) Prototypes deletion dialog.

Figure 6.6: Prototypes' dialogs. It displays a prototype description and allows changing its type or removing it.

6.2.4 Signal window

The fourth and last window, the signal window, shows the current registered equipment sensors or measurements and can be used to add, remove or edit registered signals or their descriptions. Depicted in Figure 6.8, it is composed of three elements: the common navigation menu, the registered signal table, and the signals importance pie chart.

The registered signal table (Figure 6.9) presents the current registered signals, including their id, automatically generated during registration, their name or tag, unit, and a brief description. This component allows listing 5, 10 or 25 signals at the same time, with pagination. In the future it should also allow deleting or registering new signals, but currently these operations are enabled only in the backend, to be described in Section 6.3.

The last component, shown in Figure 6.10, is the signal importance pie chart. It represents the relative importance of each signal for the current detection, making this chart useful to detect possible problems in the sensors, as a defective sensor would either gain importance or become irrelevant. The procedure to compute the signal importance is described at Section 6.3.

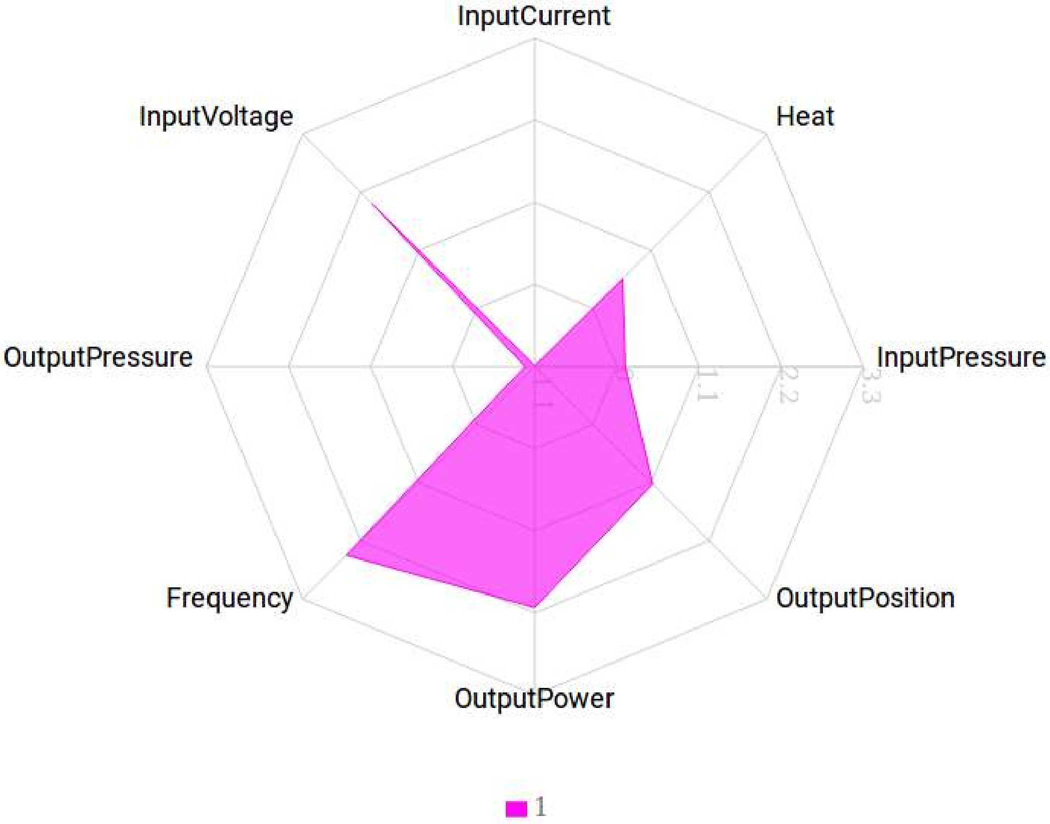
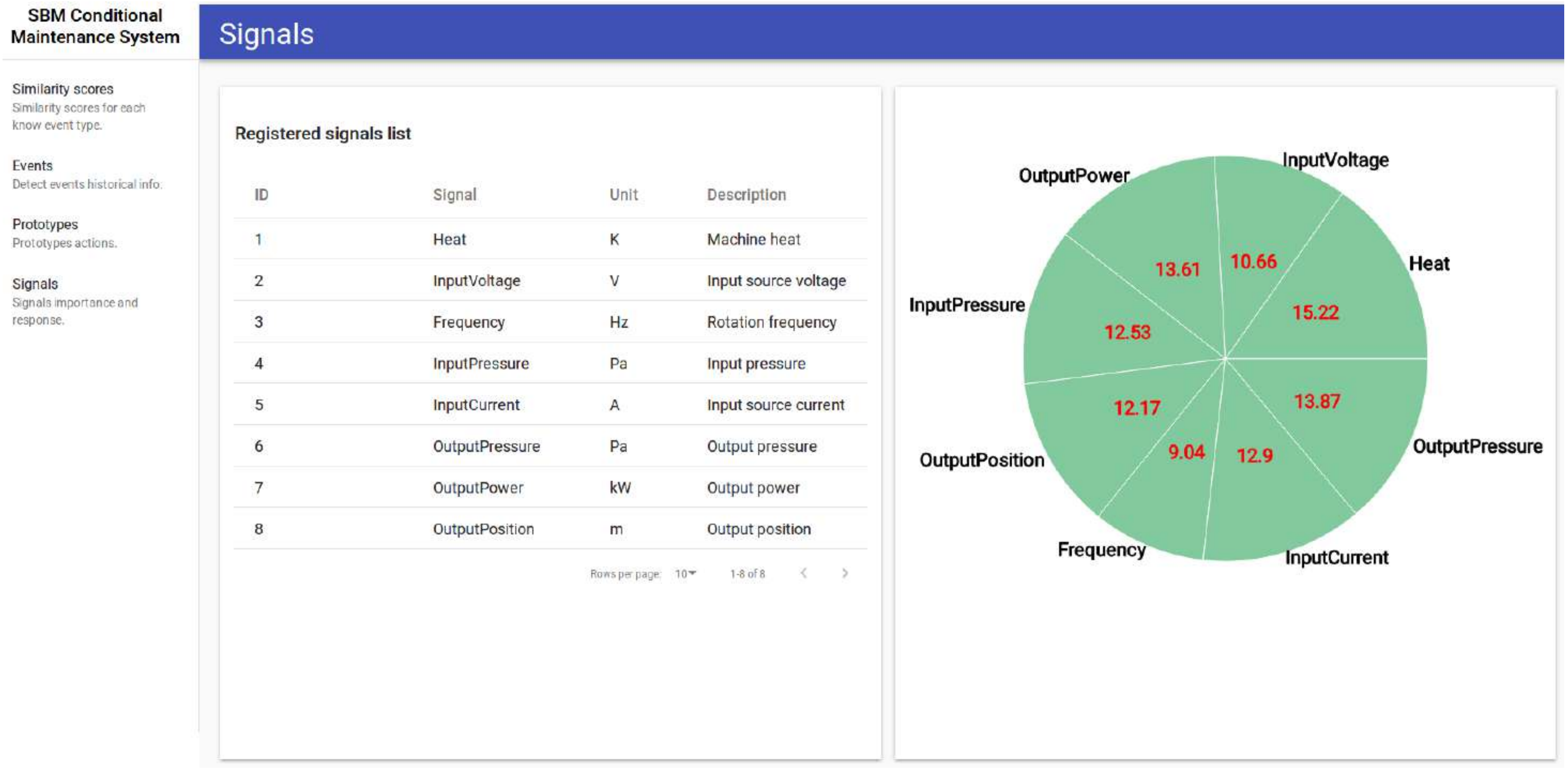


Figure 6.7: Prototype radar chart. It allows visualizing all normalized signals values for a single prototype instance.



SBM Conditional Maintenance System

Similarity scores
Similarity scores for each know event type.

Events
Detect events historical info.

Prototypes
Prototypes actions.

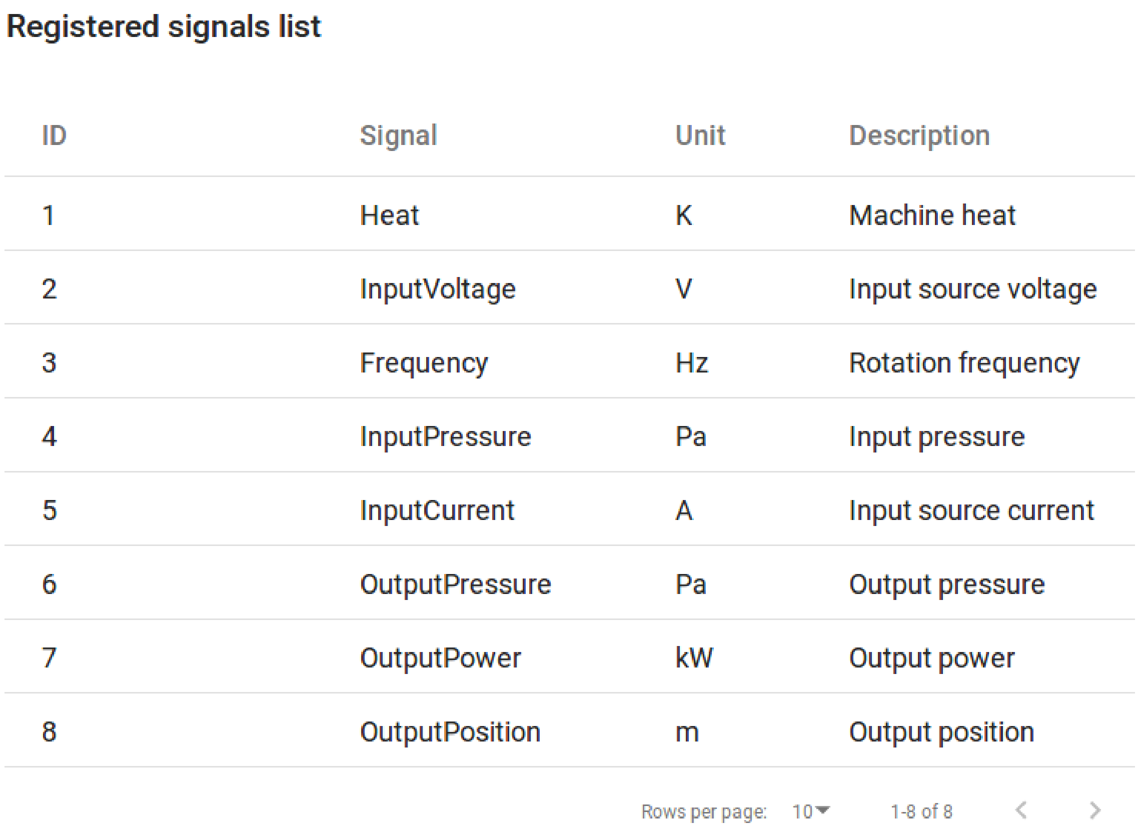
Signals
Signals importance and response.

Signals

Registered signals list

ID	Signal	Unit	Description
1	Heat	K	Machine heat
2	InputVoltage	V	Input source voltage
3	Frequency	Hz	Rotation frequency
4	InputPressure	Pa	Input pressure
5	InputCurrent	A	Input source current
6	OutputPressure	Pa	Output pressure
7	OutputPower	kW	Output power
8	OutputPosition	m	Output position

Figure 6.8: Signal window. This windows shows the list of registered signals and their current relative importance as a pie chart.



Registered signals list

ID	Signal	Unit	Description
1	Heat	K	Machine heat
2	InputVoltage	V	Input source voltage
3	Frequency	Hz	Rotation frequency
4	InputPressure	Pa	Input pressure
5	InputCurrent	A	Input source current
6	OutputPressure	Pa	Output pressure
7	OutputPower	kW	Output power
8	OutputPosition	m	Output position

Figure 6.9: Registered signal table. It presents a list of the current registered signals.

6.2.5 Web interface framework

This section briefly describes some technical details about how the web interface operates. The proposed system follows a web server approach, with a browser-based user interface as frontend which receives data from the data layer backend. The web server approach was chosen given its flexibility and interoperability between multiple operational systems and devices, making it accessible by desktop or mobile users.

As discussed above, while for an operator or a user the web server is composed of four distinct windows, in truth, it is a single window application which dynamically renders components given a different route. These components and the windows were made in JavaScript language, chosen by its popularity and the possibility of enabling dynamical elements [73, 74].

However, there are many framework and libraries to design and implement applications in JavaScript. In this work we used the React library [75] with Redux state container system [76]. These were selected given their current community support, extensive documentation, and the author familiarity with them.

To provide a consistent feeling along the user interface we used the Material UI library [77] to generate the common components (buttons, menus, text, etc.) and

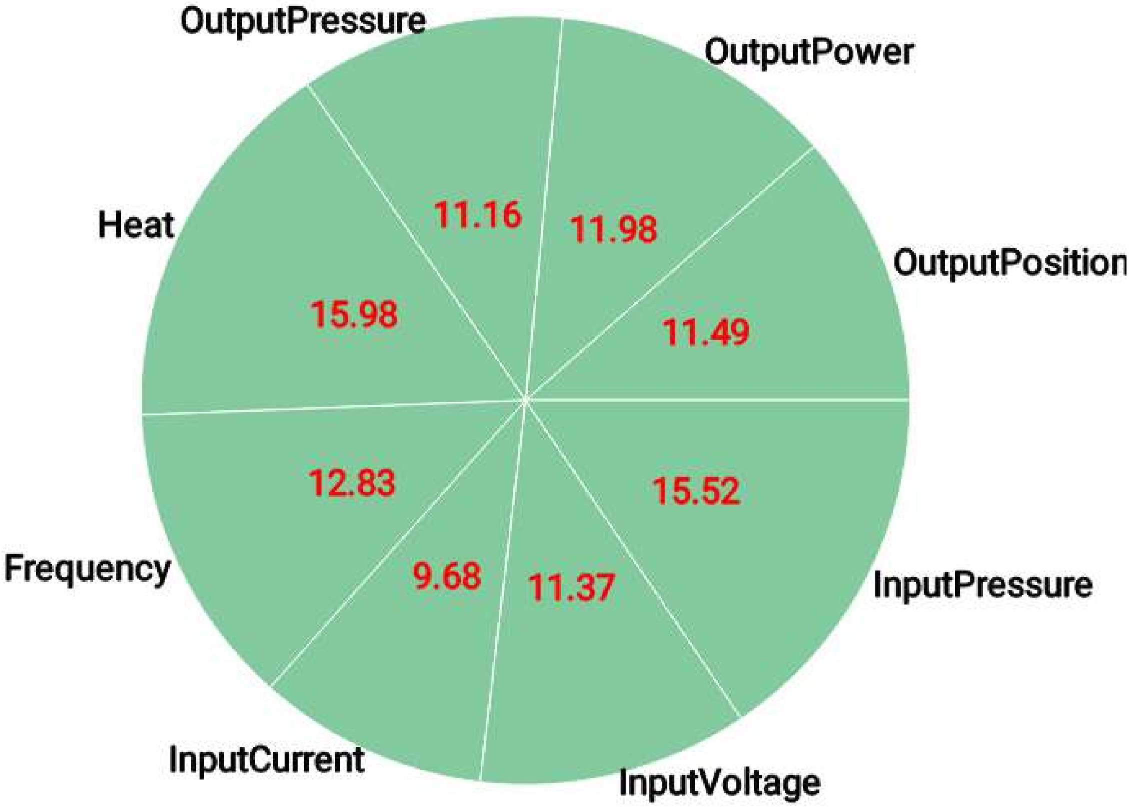


Figure 6.10: Signal importance pie chart. It shows the relative importance of each registered signal given the equipment current condition.

the Recharts library [78] to produce charts and plots for data exploration.

Lastly, to fetch data from the backend data layer, we implemented a pooling procedure which updates components by periodically querying the data layer for new data. While this approach is less than ideal, since it makes unnecessary requests when there is no change or event between two requests, given the limit time and expertise in designing such systems, it achieves the proposed objectives.

6.3 Data layer

This section presents the service data layer and backend, the proposed system powerhouse. The data layer is composed of the following:

- the *online detection algorithm*, divided into the *similarity function learning algorithm* and the *online SBM algorithm*;
- the *data access endpoints*, entry points of the service, where the communication between the data layer and external services, including the web interface, occurs;
- the *data processing block*, which process the incoming events and returns relevant information, including applying the online detection algorithm, saving events in the database, and generating auxiliary information;
- and the relational database, where all incoming and produced data are saved.

We start by describing the online detection algorithm, responsible for detecting possible outliers or faults during real-time events. This algorithm was proposed to work in the context of direct supervision of an operator and to cope with real-time sensors and equipment failures. However, we did not find any dataset with these characteristics, neither we could devise one given this work time limitations. Thus, the proposed online detection algorithm was not quantitatively assessed.

Also, while the described algorithm is used by the proposed service, any algorithm with the same properties (online learning, outliers detection, and features importance) could be used by the proposed system.

6.3.1 Similarity function learning

Learning a good distance metric is important to many applications, such as content-based image retrieval and classification in computer vision, text analysis, and others [79, 80]. As an example, $k$nn classifiers achieve a significant accuracy increase with appropriately-designed distance metric, comparing with standard Euclidean distance [80].

As discussed in Chapter 3, since the SBM framework comes from the same learning algorithms as the $k$nn classifier, it could also benefit from learning a custom distance metric directly from the relevant data.

To develop an algorithm to learn a custom metric, first we assume a binary classification problem with a dataset $\mathcal{D}$ composed of N samples cases $(\mathbf{x}_i, \mathbf{x}_j, y_{ij})$, where $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^m$ are pair of points to be compared and

$$y_{ij} = \begin{cases} +1 & \text{if the pair } \mathbf{x}_i, \mathbf{x}_j \text{ are in the same set;} \\ -1 & \text{otherwise.} \end{cases} \tag{6.1}$$

Defining the distance metric matrix as $\mathbf{A} \in \mathbb{R}^{m \times m}$, and the distance between any two points as

$$d^2_{\mathbf{A}}(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|^2_{\mathbf{A}} = (\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}} \mathbf{A} (\mathbf{x}_i - \mathbf{x}_j), \tag{6.2}$$

then, with a logistic regression model, we can compute the similarity as the probability of any two points belonging in the same set as

$$s_{ij} = \frac{1}{1 + \mathrm{e}^{y_{ij}\left[d^2_{\mathbf{A}}(\mathbf{x}_i, \mathbf{x}_j) - \mu\right]}}, \tag{6.3}$$

where parameter μ represents the distance threshold that indicates if points are in the same set.

Taking the negative log-likelihood as the loss function, we have

$$I_{\mathrm{emp}}[\mathbf{A}, \mu] = -\sum \log\left\{1 + \mathrm{e}^{y_{ij}\left[d^2_{\mathbf{A}}(\mathbf{x}_i, \mathbf{x}_j) - \mu\right]}\right\}, \tag{6.4}$$

thus, we can cast the distance metric learning problem as the optimization problem of minimizing Eq. (6.4). However, since the SBM framework starts from solving an *one class classification problem* [81, 82], parameters $\mathbf{A}$ and μ could be unbounded. As such, a regularization factor is added to restrict the possible set of solutions. The regularized version of the loss function is given by

$$I_{\mathrm{emp}}[\mathbf{A}, \mu] = -\sum \log\left\{1 + \mathrm{e}^{y_{ij}\left[d^2_{\mathbf{A}}(\mathbf{x}_i, \mathbf{x}_j) - \mu\right]}\right\} + \gamma \|\mathbf{A}\|^2_F, \tag{6.5}$$

where $\|\mathbf{A}\|^2_F = \mathrm{tr}\left(\mathbf{A}\mathbf{A}^{\mathrm{T}}\right)$ is the Frobenius norm and $\gamma \in [0, 1)$ is a regularization parameter.

While Eq. (6.5) could be solved as a positive semi-definite optimization problem [79, 80]

$$\begin{aligned} &\min_{\mathbf{A},\mu} = I_{\mathrm{emp}}[\mathbf{A}, \mu] \\ &\text{s.t. } \mathbf{A} \succeq 0, \mu \geq 0, \end{aligned} \tag{6.6}$$

in this work a stochastic gradient approach was followed to allow parameters $\mathbf{A}$ and μ to be updated in an online fashion. We can define their gradients as

$$\frac{\partial I_{\text{emp}}[\mathbf{A},\mu]}{\partial \mathbf{A}} = \sum y_{ij}\left(s_{ij}-1\right)\left(\mathbf{x}_i-\mathbf{x}_j\right)\left(\mathbf{x}_i-\mathbf{x}_j\right)^{\mathrm{T}} + \gamma 2\mathbf{A}, \tag{6.7}$$

and

$$\frac{\partial I_{\text{emp}}[\mathbf{A},\mu]}{\partial \mu} = -\sum y_{ij}\left(s_{ij}-1\right), \tag{6.8}$$

where s_{ij} is the similarity function described in Eq. (6.3). Then, we can update each parameter as

$$\mu_{n+1} = \mu_n - \eta\frac{\partial I_{\text{emp}}[\mathbf{A},\mu]}{\partial \mu}, \tag{6.9}$$

$$\mathbf{A}_{n+1} = \mathbf{A}_n - \eta\frac{\partial I_{\text{emp}}[\mathbf{A},\mu]}{\partial \mathbf{A}}, \tag{6.10}$$

where η is the learning rate parameter. As initial conditions we start with

$$\mathbf{A}_0 = \left[\frac{1}{N}\sum\left(\mathbf{x}_i-\mathbf{x}_j\right)\left(\mathbf{x}_i-\mathbf{x}_j\right)^{\mathrm{T}}\right]^{-1},$$

which is the *estimated precision matrix* [83], and $\mu_0 = 0$. This initial condition is equivalent to using the Mahalanobis distance [33, 79, 80] followed by a logistic regression. The next section introduces how this distance metric is integrated in the proposed online SBM framework.

6.3.2 Online SBM

In Chapter 3 the SBM framework was described as a supervised learning algorithm, where the possible states are known a priori and the training algorithm tries to find the best prototype set for each class. However, these conditions are not always possible during deployment, as there are differences in production environment, new unknown faulty or production conditions, and interaction with new factors, such as new equipment, processes and personnel.

To cope with these issues, the proposed system should be flexible, for possible changes in the production environment; adaptable to the different conditions of an equipment; and robust against possible perturbations. This section describes the proposed approach to add these characteristics to the original SBM framework, by devising an online version of the original algorithm, such that, with manual input, it could correctly learn and adapt for new and different operational conditions.

The proposed online algorithm is very similar to one of its offline training counterparts, the similarity threshold method, proposed in Section 3.4.2. The online learning algorithm can be divided in 5 steps:

1. *Initialization*: since the online algorithm works almost in an unsupervised fashion, it assumes that the first sample comes from a healthy state to initialize

$$\mathcal{P}_0 \leftarrow \{\mathbf{x}_0\}. \tag{6.11}$$

Therefore, when this assumption fails the operator should correct the system with the correct class;

2. *Prediction error*: Given a set of prototypes $\mathcal{P}_n$ at instant n, during the prediction step the error between the current sample $\mathbf{x}_n$ and each prototype is computed as

$$\mathbf{e}_{np} = \Big[|x_{n1} - x_{p1}| \quad |x_{n2} - x_{p2}| \quad \ldots \quad |x_{nm} - x_{pm}|\Big]^{\mathrm{T}}, \quad p \in \mathcal{P}_n; \tag{6.12}$$

3. *Instance similarity*: the prediction error $\mathbf{e}_{np}$ for each prototype is used as input for the similarity function $f_n(\bullet)$. In this case, the one described in Eq. (6.3)

$$s_{np} = f_n(\mathbf{e}_{np}); \tag{6.13}$$

4. *Predicted similarity and class*: computed in a similar fashion as its offline counterpart described in Section 3.4.1

$$g_{nc} = \prod_p s_{np}, p \in \mathcal{P}_c \tag{6.14}$$

$$c^* = \arg\max_c \{g_{nc}\}. \tag{6.15}$$

5. *Update*: Lastly, the update strategy first tries to add a new prototype when necessary, then it updates the similarity function as described in Section 6.3.1. Given the global similarity for the selected class

$$g_n = \max_c \{g_{nc}\}, \tag{6.16}$$

the prototype insertion is made using the following heuristic:

$$\mathcal{P}_n \leftarrow \mathcal{P}_{n-1} \cup \{\mathbf{x}_n\} \quad \text{if} \begin{cases} P_{n-1} < P_{\max} \\ \wedge \quad g_n \leq \tau \\ \wedge \quad g_{n-1} > g_n. \end{cases} \tag{6.17}$$

where $P_n = |\mathcal{P}_n|$ and $P_{\max}$ are the current number of prototypes and maximum allowed number of prototypes, respectively. The threshold τ is the minimum similarity and in this work is empirically set as $\tau = 0.7$.

The similarity function $\mathbf{A}_c$ and μ_c parameters are updated by computing their gradients assuming that the chosen label c^* is the correct class, thus

$$y_{npc} = \begin{cases} +1 & \text{if } c = c^*, \\ -1 & \text{otherwise.} \end{cases} \tag{6.18}$$

All these steps and some intermediate steps are depicted in Algorithm 6.1.

Algorithm 6.1 Online similarity function and prototype selection SBM.

```
function ONLINE_SBM($\mathcal{X}$, $\gamma$, $P$)
    Initialize $\mathcal{P}_0 \leftarrow \{\mathbf{x}_0\}$, $f_0$
    while $n < |\mathcal{N}|$ do
        Prototypes deviation $\mathbf{e}'_{np} = \mathbf{x}_n - \mathbf{x}_p$, $p \in \mathcal{P}$
        Element-wise absolute deviation $\mathbf{e}_{np} = (|e_{npi}|)$
        Instance similarity $s_{np} = f_n(\mathbf{e}_{np})$
        Class similarity $g_{nc} = \prod_p s_{np}, p \in \mathcal{P}_c$
        while $|\mathcal{P}_n| \leq P$ do
            $\mathcal{P}_n \leftarrow$ PROTOTYPE_INSERTION($\mathcal{P}_n$, $\mathbf{x}_n$)
            $f_n \leftarrow$ UPDATE_SIMILARITY($f_n$, $\mathbf{x}_n$)
        end while
    end while
end function
```

6.3.3 Data access endpoints

Endpoints are the entry points of the service. In our case, they are channels where the sensor data is received and the information produced by the system can be consulted. However, while accessible, the information provided at each endpoint is not presented in an easy or even intelligible way. Thus, the necessity of a user interface, such as the one described in Section 6.2.

To make the proposed system compatible with other services, it follows the Representational State Transfer (REST) architectural style [84], accepting requests operations such as GET, POST and DELETE [1]. Since the author learned about REST interfaces during the creation of this thesis, this work does not make any claim that the application is completely REST compliant. A brief description of each endpoint follows.

[1] These request operations follow the Hypertext Transfer Protocol (HTTP) methods. The GET method lists a collection or retrieves an element given an endpoint, the POST request creates a new entry in a collection, and the DELETE method removes an element or collection. Their usage is described as each endpoint is presented.

Signals

The signal endpoint "/SIGNALS" returns or receives information in two formats. When accessed directly from the main endpoint by a GET operation, it returns the list of signals. This is used by the web interface to present the registered signals list (Figure 6.9). When the request is a POST, it registers the incoming data as a new signal. The system requires that the first requests are POST requests to this endpoint, to register the equipment sensors.

The second format, "/SIGNALS/ID_NUMBER" returns a single signal with the id ID_NUMBER when the request is a GET, or removes a registered signal when the request is a DELETE.

Values

This endpoint receives and returns the equipment sensor values at each instant. It is the main endpoint of the system and, with the signal endpoint, the only two endpoints that receive incoming data.

It behaves in a similar fashion as the signal endpoint. The "/VALUES" endpoint accept GET requests, returning the list of historical values, or POST, appending a new value to the list. This operation has a restriction that only registered sensors are saved. A POST request with unregistered sensors will be ignored or result in error. A GET request to "/VALUES/ID_NUMBER" returns the sensors at a single instant given the ID_NUMBER. DELETE requests are not accepted, as these values are historical data and do not allow overwritten operations.

Events

The first endpoint that returns generated data. It represents registered events, such as normal, faulty, and anomalous events, and always starts with two valid events: the *default event*, based on the equipment state when the fault detection system is initialized; and the *unknown event*, which represents any detected event that was not identified. The remaining events are supplied by the operator using the web interface (Figure 6.3).

As with the previous endpoint, it can be accessed in two formats. The "/EVENTS" returns a list of events (GET) or creates a new event type (POST). POST requests should only be made by the web interface, not directly by an operator or any external systems.

The second format, "/EVENTS/ID_NUMBER", returns the event type with its ID_NUMBER when a GET request is received. Remaining request types are ignored.

Prototypes

This endpoint lists the current set of prototypes, including their values and event type. The "/PROTOTYPES" format lists (GET) the prototypes. POST requests are not accepted, as the only way of generating a new prototype is internal, as a byproduct of Algorithm 6.1.

The second format, "/PROTOTYPES/ID_NUMBER", returns a single prototype given their ID_NUMBER with a GET, deletes a prototype with a DELETE, or modifies a prototype type with a PUT. These operations should not be used directly, only by means of the web interface edition and deleting menus (Figure 6.6).

Health scores

The endpoint "/HEALTH-SCORES" returns Algorithm 6.1 global scores output for each class at each instant. It is computed when a new value arrives at the "/VALUES" endpoint. This endpoint only accepts GET requests, either retuning the list of health scores at each instant (/HEALTH-SCORES) or the scores of a single instant (/HEALTH-SCORES/ID_NUMBER). An example of usage of this endpoint is the health scores component shown in Figure 6.1.

Importance

As with the health scores, this endpoint values are computed when new data arrives at the "/VALUES" endpoint. It returns importance scores for each sensor given the input values.

In this work we chose an approach similar to the *permutation importance* used in random forests [59]: given the current sample $\mathbf{x}_n$, we produce, for each prototype $\mathbf{x}_p$ and error $\mathbf{e}_{np}$, a perturbed version where a single measurement at m is permuted from its original value with random Gaussian noise σ_m, producing a new error

$$\hat{\mathbf{e}}_{np}^{m} = \mathbf{e}_{np} + \mathbb{1}_m \sigma_m, \tag{6.19}$$

where $\mathbb{1}_m$ denotes the vector with a 1 in the m-th coordinate and 0's elsewhere. Then, for each measurement m, we can compute the deviation between the original g_{nc} and the perturbed g_{nc}^{m} as

$$\widehat{\text{imp}}_{mc} = \left| g_{nc} - g_{nc}^{m} \right|, \tag{6.20}$$

and the signal relative importance as

$$\text{imp}_{mc} = \frac{\widehat{\text{imp}}_{mc}}{\sum_{kc} \widehat{\text{imp}}_{kc}}. \tag{6.21}$$

As with the health score, the "/IMPORTANCE" only accepts GET requests, returning a list or the importance scores at a given instant (/IMPORTANCE/ID_NUMBER). The importance component, shown in Figure 6.10, feeds from this endpoint to obtain information.

Detected events

The events detected by the proposed system are exposed by this endpoint. It returns the event start and end times, its type, and id. Like the previous elements, it only accepts GET requests. This endpoint generates event data used in the event dialog (Figure6.3).

Health status

This a special endpoint which returns a single value, the current health score computed as the mean score considering all distinct normal events. The HEALTH-STATUS only receives GET requests. Its main usage is to generate alarms when an anomalous or faulty condition occurs.

Detected event scores

The DETECTED-EVENT-SCORE/ID_NUMBER complements the detected events endpoint, returning only health score data during the interval of a chosen event. This data is used to generate the events charts.

6.3.4 Data processing and storage flow

This section presents how the data processing and storage occurs in the proposed framework.

Figure 6.11 is a succinct description of the event flow when a new signal set comes from the equipment. All processes start when a new set of signals is received at the signal endpoint. If this set is the first sample set and there is no predefined prototype, this first sample becomes a prototype with normal state. Otherwise, it passes through the normal detection steps following Algorithm 6.1. First, the system computes their deviation against the current set of prototypes. Deviation values are then used to compute the global similarity and instance similarity. These values are also used to compute the marginal importance, which is the importance of each sensor or signal, as described in Section 6.3.3.

The input data and the output of these tasks are saved in the relational database to be available for future use, analysis, or audition. These are then used to update the detector and, if necessary, to add a new prototype or flag an anomalous event.

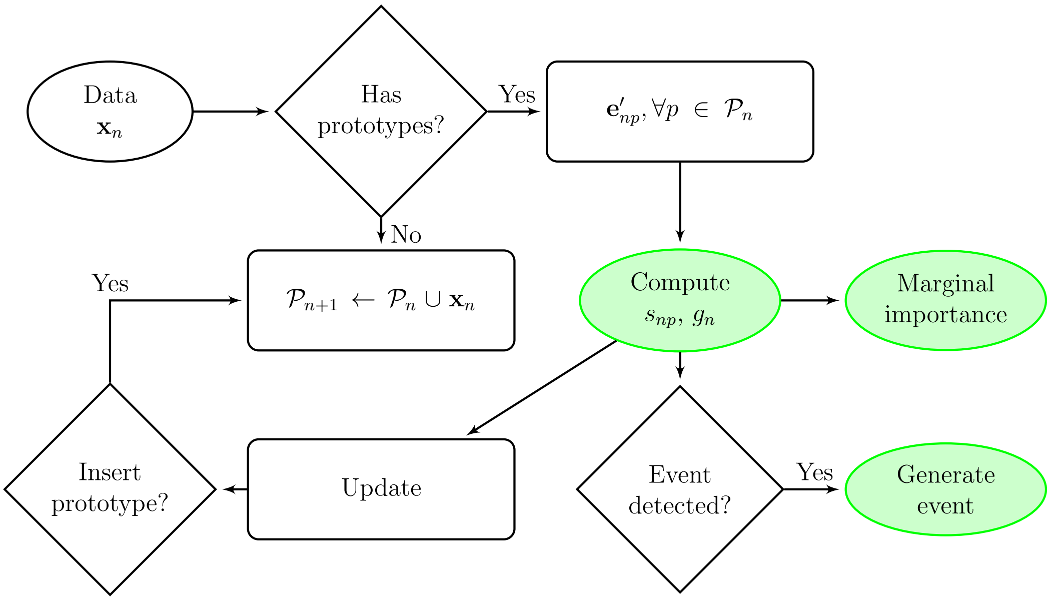


Figure 6.11: Web application data processing event flow.

The relational database is where all the relevant received or produced data are saved. The database server is a PostgreSQL [85] database management system, chosen given its robustness, flexibility, and ease of use. To simulate a deployment environment, the database and the remaining service components, the web interface and the data layer, were "*containerized*" [2], allowing a fast deployment and isolating the development environment from the application.

The database can be decomposed in the following models: signals, signal values, events, prototypes, detected events, health scores, health scores per prototype, similarity function coefficients, and the signal marginal importance. All models have four common fields: *created*, *updated*, which hold the creation and update time stamps, respectively; an *etag*, for cache generation; and a unique identification number. The other fields of each model are:

- *signals*: sensor model, it represents the registered sensors. Composed of fields *name*, *unit* (SI units), *description*, and if it is *active* signal (not deleted);
- *signal values*: signal *values* are JavaScript Object Notation (JSON) stored fields in a key-value fashion. Each entry is saved with its respective *timestamp*;
- *events*: saved event type with their *names*, *description*, and *base type* (*healthy*, *shutdown*, *faulty*, and *unknown*);
- *prototypes*: represents the selected prototypes, with JSON field *values*, its *timestamp*, and its relationship with an event type being represented by an

[2] An application is *containerized* when its running in an isolated virtual container environment, similar to a virtual machine but with less overhead. In this work we chose Docker [86] as our container implementation as Docker containers are becoming the "standard" containerization solution currently.

event id;

- *detected events*: represents a *detected event*, include its *event type id*, its *start* and *end* timestamp;
- *health score*: represents the computed health score for each known class at a given *timestamp*, with their *scores* and *event* types as JSON maps;
- *prototype scores*: scores per prototype, with fields *timestamp* and *scores*, a JSON map with the prototypes ids as keys;
- *signals marginal importance*: the computed signal importance *values* are stored as a JSON field, and their *timestamp*;
- *coefficients*: computed coefficients *values* for each *event id*, saved as JSON.

These models are used by the service to store, retrieve, and provide the information for each endpoint. While this information should not be accessed directly, a user could access and query the production database to query the service data. The models and the backend were made with Flask [87], an extremely popular web framework written in Python [88], language which the author has familiarity.

6.4 Conclusion

This chapter presents some small contributions to this work, including:

- It extends the original SBM framework with an online alternative, with the ability of learning the best similarity function and heuristics to define new events, classes, and prototypes;

- It presents a web application based on the SBM framework, to enable an operator to detect and monitor anomalous states of an equipment in real-time by providing relevant information, automating decisions, or helping with the operators decision process;

- It presents a data processing framework, which feeds the web application and processes the incoming signals. In this work this framework was used with the SBM technique, but it is flexible enough to be used by other outlier or fault detection approaches.

While there is still room for improvement, we hope that the current approach could be used in production environment without any or with minimal changes. The next chapter concludes this work and presents future directions for this framework.

Chapter 7

Conclusion

7.1 Discussion

In this work we proposed a data-driven condition-based maintenance (CBM) system based on similarity-based modeling (SBM) for automatic machinery fault diagnosis. The proposed system provides information about the current state, degree of the anomaly, and returns sets of exemplars that can be employed to describe the machine state in a sparse fashion, which can be examined by the operator to assess the decision. The system is modular and data-agnostic, enabling its use in different equipment and data sources with small modifications. The system was evaluated using three databases: a comprehensive rotating-machinery database with multiple faults referred to as MaFaulDa [25]; the CWRU [6] bearing database, the current standard database for bearing fault diagnosis; and an oil-platform injection pump system dataset, consisting of multivariate time-series from oil-platform equipment.

One of the main contributions of this work was the extensive study of the use of multiclass SBM on different databases. Other contributions regard novel methods for selecting prototypes for the SBM models and the use of new similarity metrics. These contributions achieved the goal of increasing the SBM performance in a fault classification scenario while reducing its computational complexity. The usage of SBM either as a stand-alone classifier or as a feature generator for off-the-shelf classifiers has also been investigated. We also compared the proposed approach against more traditional prototype selection methods, such as the methods used in $k$nn classifier, *condensation* and *edition* [29], in the SBM framework. This procedure not only gave us some insight about the desired characteristics of a prototype detection method, but also asserted the robustness of the proposed approaches, which achieved better or equivalent results in each comparison.

The extensive analysis of different approaches and parameters of the SBM system, and the obtained results in the MaFaulDa and CWRU databases show that the proposed modifications produced a new multiclass classifier that can be used as a standalone classifier, with acceptable to good performance, or as a feature generator for a downstream classifier, achieving state of the art results in each dataset, while reducing its original formulation computational complexity. These results, presented in Chapters 4, indicate that the proposed approach based on SBM is robust and worth further investigation.

Chapter 5 presents a real environment application of the proposed framework in a temporal series, including an exploratory analysis, data description, and the usage of the SBM as a fault detector. The results demonstrated that, while the proposed approach is robust in a great range of parameters, corroborating with the previous findings, it suffers from the lack of temporal consistency between each decision. This indicates that the method could benefit from some temporal consistency.

Lastly, Chapter 6 presents a production ready version of the proposed framework, including a graphical interface, its underlying web server, and an adaptation of the SBM framework for real-time detection, with online prototypes and metric learning. While online metric learning and prototype selection could not be explored in depth, missing qualitative and quantitative results, they present another new research area in similarity-based modeling still unexplored, and could solve the temporal consistency shortcomings found in the offline SBM.

7.2 Future work

There are at least three paths in the future investigation. The first path comprehends further investigation of similarity methods. This work is still in development and, while the proposed contributions produced clear performance gains when compared with the original SBM model, there is still room for improvement, within the SBM and other similarity approaches. Some topics aligned with this path include: to study other methods for prototype selection for similarity methods; to propose new distance metrics and similarity functions to improve the performance of similarity models [67]; and to study and compare the SBM methodology with other prototype-based and similarity-based methodologies. As an example, using denoising autoencoders as monitor models, following the approach presented in [89].

As part of this first approach, assuming that real failure and fault states live in a low-dimension manifold, a metric that selects the best set of features to explore this space would have tremendous advantage against heuristic-based metrics [79, 80]. Learning similarity functions could provide an important tool to produce discriminative models and produce interesting insights of the nature of the data, including feature selection and importance measurements [90]. While in this work we made some small progress in this direction, there still a lot to explore, including producing qualitative and quantitative results with this approach in different datasets.

The second path consists of adding another block in the proposed system, the *prognosis* block. As described in [2], the CBM system final result is an estimate of *when* a failure will occur. This estimate of the remaining useful life (RUL) must precede the failure with time for the prescribed maintenance action, eliminating unexpected breakdowns and maximizing the equipment life time [2]. However, the current approach does not exploit the natural consistency of a time series, evaluating each state as episodic. As such, other approaches, which can use the temporal information, can be sought to be used along SBM models or as a substitute, if deemed necessary.

Lastly, the third step explores the original usage of the SBM framework as a novelty/outlier detector. The current models do not provide the necessary discriminative power for some high precision tasks nor learn the best similarity metric for a functional condition state. This previous remark about the discriminative power of the current models invites two questions: “is it really possible to increase the discriminative power of SBM models?” and, given that the previous question answer is “yes”, “how can we achieve the necessary discriminative power?”. The study of the original *one-class* formulation of SBM models, and their usage as *novelty* or *outlier* detectors [82] should provide the necessary answers to these questions. As such, another possible area for future work includes applying what we learned about the SBM as multiclass classifier back into the original formulation and compare it against other outliers and novelty detectors, such as the ones discussed in [82], under different databases [91, 92].

Bibliography

[1] COBLE, J. B. *Merging Data Sources to Predict Remaining Useful Life – an Automated Method to Identify Prognostic Parameters.* Ph.D. Thesis, University of Tennessee, 2010.

[2] PALEM, G. "Condition-based Maintenance Using Sensor Arrays and Telematics", *International Journal of Mobile Network Communications & Telematics*, v. 3, n. 3, pp. 19–28, 2013.

[3] SINGER, R. M., GROSS, K. C., HERZOG, J. P., et al. "Model-based Nuclear Power Plant Monitoring and Fault Detection: Theoretical Foundations". In: *Proc. International Conference on Intelligent Systems Applications to Power Systems*, July 1997.

[4] BIEN, J., TIBSHIRANI, R. "Prototype Selection for Interpretable Classification", *The Annals of Applied Statistics*, v. 5, n. 4, pp. 2403–2424, 2011.

[5] PESTANA-VIANA, D., ZAMBRANO-LÓPEZ, R., DE LIMA, A. A., et al. "The Influence of Feature Vector on the Classification of Mechanical Faults Using Neural Networks". In: *Proc. Latin American Symposium on Circuits and Systems*, 2016.

[6] LOPARO, K. A. "Bearings Vibration Data Set, Case Western Reserve University". `http://csegroups.case.edu/bearingdatacenter/home`, 2003. accessed November 25, 2016.

[7] ISO 10303-226:2014(E). *Automation Systems and Integration – Product Data Representation and Exchange.* Standard, International Organization for Standardization, Geneva, CH, March 2014.

[8] HERZOG, J. P., GANDHI, D., NIEMAN, B. *Making Decisions About the Best Technology to Implement.* Technical report, Smartsignal/GE Intelligent Platforms, 2011.

[9] JARDINE, A. K. S., LIN, D., BANJEVIC, D. "A Review on Machinery Diagnostics and Prognostics Implementing Condition-based Maintenance",

Mechanical Systems and Signal Processing, v. 20, n. 7, pp. 1483–1510, 2006.

[10] LOUPPE, G. *Understanding Random Forests: From Theory to Practice.* Ph.D. Thesis, University of Liege, Belgium, 2014.

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

[12] HASTIE, T., TIBSHIRANI, R., FRIEDMAN, J. *The Elements of Statistical Learning.* New York, NY, USA, Springer New York, 2001.

[13] VAN DER HEIJDEN, F., DUIN, R. P. W., DE RIDDER, D., et al. *Classification, Parameter Estimation and State Estimation.* UK, John Wiley & Sons, 2005.

[14] RIFKIN, R., YEO, G., POGGIO, T. "Regularized Least Squares Classification". In: Suykens, Horvath, Basu, et al. (Eds.), *Advances in Learning Theory: Methods, Model and Applications*, v. 190, *NATO Science Series III: Computer and Systems Sciences*, VIOS Press, chap. 7, pp. 131–154, 2003.

[15] SCHÖLKOPF, B., HERBRICH, R., SMOLA, A. J. "A Generalized Representer Theorem". In: *Proc. 14th Annual Conference on Computational Learning Theory and 5th European Conference on Computational Learning Theory*, pp. 416–426. Springer-Verlag, 2001.

[16] POWERS, D. M. W. *Evaluation: from Precision, Recall and F-factor to ROC, Informedness, Markedness and Correlation.* Technical report, Flinders University, Adelaide, Australia, 2007.

[17] LANDGREBE, T. C. W., DUIN, R. P. W. "Efficient Multiclass ROC Approximation by Decomposition via Confusion Matrix Perturbation Analysis", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 30, n. 5, pp. 810–822, 2008.

[18] CHAI, T., DRAXLER, R. R. "Root Mean Square Error (RMSE) or Mean Absolute Error (MAE)? – Arguments Against Avoiding RMSE in the Literature", *Geoscientific Model Development*, v. 7, n. 3, pp. 1247–1250, 2014.

[19] KUTNER, M., NACHTSHEIM, C., NETER, J. *Applied Linear Regression Models.* Chicago, IL, McGraw-Hill Higher Education, 2003.

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

[21] OLIPHANT, T. E. “Python for Scientific Computing”, *Computing in Science and Engineering*, v. 9, n. 3, pp. 10–20, 2007.

[22] VAN DER WALT, S., COLBERT, S. C., VAROQUAUX, G. “The NumPy Array: A Structure for Efficient Numerical Computation”, *Computing in Science Engineering*, v. 13, n. 2, pp. 22–30, 2011.

[23] LI, P., KONG, F., HE, Q., et al. “Multiscale Slope Feature Extraction For Rotating Machinery Fault Diagnosis Using Wavelet Analysis”, *Measurement*, v. 46, n. 19, pp. 497–505, 2013.

[24] LIU, J., WANG, W., GOLNARAGHI, F. “An Enhanced Diagnostic Scheme for Bearing Condition Monitoring”, *IEEE Transactions on Instrumentation and Measurement*, v. 59, n. 2, pp. 309–321, 2010.

[25] “MaFaulDa - Machinery Fault Database”. `http://www02.smt.ufrj.br/~offshore/mfs/`, 2016. accessed November 22, 2016.

[26] DE LIMA, A. A., PREGO, T. M., NETTO, S. L., et al. “On Fault Classification in Rotating Machines Using Fourier Domain Features and Neural Networks”. In: *Proc. Latin American Symposium on Circuits and Systems*, 2013.

[27] BOUDIAF, A., MOUSSAOUI, A., DAHANE, A., et al. “A Comparative Study of Various Methods of Bearing Faults Diagnosis Using the Case Western Reserve University Data”, *Journal of Failure Analysis and Prevention*, v. 16, n. 2, pp. 271–284, 2016.

[28] SMITH, W. A., RANDALL, R. B. “Rolling Element Bearing Diagnostics Using the Case Western Reserve University Data: A Benchmark Study”, *Mechanical Systems and Signal Processing*, v. 64-65, pp. 100–131, 2015.

[29] GARCIA, S., DERRAC, J., CANO, J., et al. “Prototype Selection for Nearest Neighbor Classification: Taxonomy and Empirical Study”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 34, n. 3, pp. 417–435, 2012.

[30] MITCHELL, T. M. *Machine Learning*. New York, NY, USA, McGraw-Hill, Inc., 1997.

[31] KONONENKO, I., KUKAR, M. *Machine Learning and Data Mining.* Chichester, UK, Horwood Publishing, 2007.

[32] GRUDZIŃSKI, K., DUCH, W. "SBL-PM: A Simple Algorithm for Selection of Reference Instances in Similarity Based Methods". In: *Proc. Intelligent Information Systems*, 2000.

[33] GOLUB, G. H., LOAN, C. F. V. *Matrix Computations.* 3rd ed. Baltimore, MD, USA, Johns Hopkins University Press, 1996.

[34] MICCHELLI, C. A. "Interpolation of Scattered Data: Distance matrices and Conditionally Positive Definite Functions", *Constructive Approximation*, v. 2, n. 1, pp. 11–22, 1986.

[35] BASAK, J. "A Least Square Kernel Machine with Box Constraints". In: *Proc. International Conference on Pattern Recognition*, December 2008.

[36] WEGERICH, S. W., WILKS, A. D., PIPKE, R. M. "Nonparametric Modeling of Vibration Signal Features for Equipment Health Monitoring". In: *Proc. IEEE Aerospace Conference*, v. 7, 2003.

[37] GENTON, M. G. "Classes of Kernels for Machine Learning: a Statistics Perspective", *Journal of Machine Learning Research*, v. 2, pp. 299–312, 2002.

[38] WEGERICH, S. W. "Similarity-based Modeling of Time Synchronous Averaged Vibration Signals for Machinery Health Monitoring". In: *Proc. IEEE Aerospace Conference*, v. 6, 2004.

[39] WEGERICH, S. W. "Similarity Based Modeling of Vibration Features for Fault Detection and Identification", *Sensor Review*, v. 25, n. 2, pp. 114–122, 2005.

[40] MOTT, J., PIPKE, M. "Similarity-based Modeling of Aircraft Flight Paths". In: *Proc. IEEE Aerospace Conference*, v. 3, 2004.

[41] TOBAR, F. A., YACHER, L., PAREDES, R., et al. "Anomaly Detection in Power Generation Plants Using Similarity-based Modeling and Multivariate Analysis". In: *Proc. American Control Conference*, v. 3, 2011.

[42] GARVEY, J., GARVEY, D., SEIBERT, R., et al. "Validation of On-line Monitoring Techniques to Nuclear Plant Data", *Nuclear Engineering and Technology*, v. 39, n. 2, pp. 133–142, 2006.

[43] HERZOG, J. P., WEGERICH, S. W., GROSS, K. C., et al. “MSET Modeling of Crystal River-3 Venturi Flow Meters”. In: *Proc. International Conference on Nuclear Engineering*, 1998.

[44] GUO, P., BAI, N. “Wind Turbine Gearbox Condition Monitoring with AAKR and Moving Window Statistic Methods”, *Energies*, v. 4, n. 11, pp. 2077–2093, 2011.

[45] DUCH, W. “Similarity-based Methods: a General Framework for Classification, Approximation and Association”, *Control and Cybernetics*, v. 29, n. 4, pp. 1–30, 2000.

[46] JANKOWSKI, N., GROCHOWSKI, M. “Comparison of Instances Selection Algorithms I. Algorithms Survey”. In: *Proc. International Conference Artificial Intelligence and Soft Computing*, pp. 598–603, Berlin, Heidelberg, Springer Berlin Heidelberg, 2004.

[47] DEVIJVER, P. A., KITTLER, J. *Pattern Recognition: A Statistical Approach.* London, UK, Prentice-Hall, 1982.

[48] CARDOT, H., CÉNAC, P., ZITT, P.-A. “Efficient and Fast Estimation of the Geometric Median in Hilbert Spaces with an Averaged Stochastic Gradient Algorithm”, *Bernoulli*, v. 19, n. 1, pp. 18–43, 2013.

[49] MARCHETTE, D. “Class Cover Catch Digraphs”, *WIREs Comput. Stat.*, v. 2, n. 2, pp. 171–177, 2010.

[50] KÖNEMANN, J., PAREKH, O., SEGEV, D. “A Unified Approach to Approximating Partial Covering Problems”, *Algorithmica*, v. 59, n. 4, pp. 489–509, 2011.

[51] RANDALL, R. B., ANTONI, J. “Rolling Element Bearing Diagnostics – A Tutorial”, *Mechanical Systems and Signal Processing*, v. 25, n. 2, pp. 485–520, 2011.

[52] ZHOU, J. H., WEE, L., ZHONG, Z. W. “A Knowledge Base System for Rotary Equipment Fault Detection and Diagnosis”. In: *Proc. Control Automation Robotics Vision*, 2010.

[53] RAUBER, T. W., BOLDT, F. D., VAREJÃO, F. M. “Heterogeneous Feature Models and Feature Selection Applied to Bearing Fault Diagnosis”, *Mechanical Systems and Signal Processing*, v. 62, n. 1, pp. 637–646, 2015.

[54] YANG, B., HAN, T., AN, J. "ART–KOHONEN Neural Network for Fault Diagnosis of Rotating Machinery", *Mechanical Systems and Signal Processing*, v. 18, n. 3, pp. 645–657, 2004.

[55] LI, B., ZHANG, P., LIU, D., et al. "Feature Extraction for Rolling Element Bearing Fault Diagnosis Utilizing Generalized S Transform and Two-dimensional Non-negative Matrix Factorization", *Journal of Sound and Vibration*, v. 330, n. 10, pp. 2388–2399, 2011.

[56] WU, S.-D., WU, P.-H., WU, C.-W., et al. "Bearing Fault Diagnosis Based on Multiscale Permutation Entropy and Support Vector Machine", *Entropy*, v. 14, n. 8, pp. 1343–1356, 2012.

[57] LI, Y., WANG, X., WU, J. "Fault Diagnosis of Rolling Bearing Based on Permutation Entropy and Extreme Learning Machine". In: *Proc. Chinese Control and Decision Conference*, 2016.

[58] "SpectraQuest, Inc." `http://www.http://spectraquest.com/`, 2016. accessed November 27, 2016.

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

[60] HART, P. E. "The Condensed Nearest Neighbor Rule", *IEEE Transactions on Information Theory*, v. 14, n. 3, pp. 515–516, 1968.

[61] WILSON, D. L. "Asymptotic Properties of Nearest Neighbor Rules Using Edited Data", *IEEE Transactions on Systems, Man, and Cybernetics*, v. 2, n. 3, pp. 408–421, 1972.

[62] VINNEM, J.-E. "Lessons from Macondo Accident". In: *Offshore Risk Assessment vol 1.: Principles, Modelling and Applications of QRA Studies*, pp. 165–177, London, UK, Springer London, 2014.

[63] TUKEY, J. W. *Exploratory Data Analysis.* Reading, MA, Addison-Wesley, 1977.

[64] CASELLA, G., BERGER, R. *Statistical Inference.* 2nd ed. Pacific Grove, CA, USA, Duxbury Press, 2001.

[65] MEINSHAUSEN, N., BÜHLMANN, P. "High Dimensional Graphs and Variable Selection with the Lasso", *The Annals of Statistics*, v. 34, n. 3, pp. 1436–1462, 2006.

[66] BECKER, H. *A Survey of Correlation Clustering.* Technical report, Columbia University, 2005.

[67] DEZA, M. M., DEZA, E. *Encyclopedia of Distances.* Berlin, Heidelberg, Springer Berlin Heidelberg, 2009.

[68] ZIMEK, A. *Correlation Clustering.* D.Sc. Thesis, Ludwig-Maximilians-Universität München, 2008.

[69] MURTAGH, F., LEGENDRE, P. "Ward's Hierarchical Agglomerative Clustering Method: Which Algorithms Implement Ward's Criterion?" *Journal of Classification*, v. 31, n. 3, pp. 274–295, 2014.

[70] ENDERS, W. *Applied Econometric Time Series.* New York, NY, USA, Wiley, 1995.

[71] BERGMEIR, C., BENÍTEZ, J. M. "On the Use of Cross-validation for Time Series Predictor Evaluation", *Information Sciences*, v. 191, pp. 192–213, 2012.

[72] KOLENCE, K. W. "The Software Empiricist", *ACM SIGMETRICS Performance Evaluation Review*, v. 2, n. 2, pp. 31–36, 1973.

[73] SEVERANCE, C. "JavaScript: Designing a Language in 10 Days", *Computer*, v. 45, pp. 7–8, 2011.

[74] FLANAGAN, D. *JavaScript: The Definitive Guide.* 6th ed. , O'Reilly Media, Inc., 2011.

[75] "React". `https://reactjs.org/`, 2018. accessed May 13, 2018.

[76] "Redux". `https://redux.js.org/`, 2018. accessed May 13, 2018.

[77] "Material-UI". `https://github.com/mui-org/material-ui/`, 2018. accessed May 13, 2018.

[78] "Recharts". `https://github.com/recharts/recharts/`, 2018. accessed May 13, 2018.

[79] KULIS, B. "Metric Learning: A Survey", *Foundations and Trends in Machine Learning*, v. 5, n. 4, pp. 287–364, 2013.

[80] YANG, L., JIN, R. *Distance Metric Learning: A Comprehensive Survey.* Technical report, Michigan State Universiy, 2006.

[81] TAX, D. M. J. *One-Class Classifications: Concept Learning in the Absence of Counter-examples*. Ph.D. Thesis, Technische Universiteit Delft, Netherlands, 2001.

[82] PIMENTEL, M. A., CLIFTON, D. A., CLIFTON, L., et al. "A Review of Novelty Detection", *Signal Processing*, v. 99, pp. 215–249, 2014.

[83] DODGE, Y. *The Oxford Dictionary of Statistical Terms*. 6th ed. Oxford, UK, Oxford University Press, 2006.

[84] FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. Ph.D. Thesis, University of California, Irvine, 2000.

[85] STONEBRAKER, M., ROWE, L. A. "The Design of POSTGRES". In: *Proc. ACM SIGMOD International Conference on Management of Data*, June 1986.

[86] "Docker". `https://www.docker.com/`, 2018. accessed May 13, 2018.

[87] "Flask". `http://flask.pocoo.org/`, 2018. accessed May 19, 2018.

[88] ROSSUM, G. V. "Python for Unix/C Programmers". In: *Proc. of the NLUUG najaarsconferentie. Dutch UNIX users group*, 1993.

[89] TAGAWA, T., TADOKORO, Y., YAIRI, T. "Structured Denoising Autoencoder for Fault Detection and Analysis". In: *Proc. Asian Conference on Machine Learning*, v. 39, 2014.

[90] BELLET, A., HABRARD, A., SEBBAN, M. "A Survey on Metric Learning for Feature Vectors and Structured Data", *CoRR*, v. abs/1306.6709, pp. 287–364, 2013.

[91] CAMPOS, G. O., ZIMEK, A., SANDER, J., et al. "On the Evaluation of Unsupervised Outlier Detection: Measures, Datasets, and an Empirical Study", *Data Mining and Knowledge Discovery*, v. 30, n. 4, pp. 891–927, 2016.

[92] POZZOLO, A. D., CAELEN, O., JOHNSON, R. A., et al. "Calibrating Probability with Undersampling for Unbalanced Classification". In: *Proc. Symposium Series on Computational Intelligence*, 2015.

Appendix A

Conditional Maintenance System Manual

A.1 Introduction

This is a user manual for the web service. It should be a reference for the current system, allowing a fast understanding of the interface and its functionalities. The application, name "*SBM Conditional Maintenance System*" or "*SBM Learner*" is a web-application made in Flask [87] and React [75], running at ports 8000 (backend) and 3000 (web frontend) of the host machine. These values are configurable and, if you are using Docker [86], you can also launch the service in your local host.

This manual is divided by sections where each section introduces a window and its functionality, in a step-by-step manner. Currently, the proposed system has four windows: the *similarity score* window, presenting historical scores for each type of event; the *event* window, where each detected event can be reviewed and managed; the *prototype* window, which allows visualizing, editing, and deleting prototypes; and the *signal* window, describing the registered signals and their importance. Each window has it own path in the application path (`http://<host-address>:3000/` [1]), as described in Table A.1.

Table A.1: Paths (url) for each window in the application.

Window	Path
Similarity scores	`http://<host-address>:3000/`
Events	`http://<host-address>:3000/events`
Prototypes	`http://<host-address>:3000/prototypes`
Signals	`http://<host-address>:3000/signals`

[1] "<host-address>" is a placeholder for the host machine address. For example, if testing the application using Docker, it should be the local host address `http://localhost:3000/`.

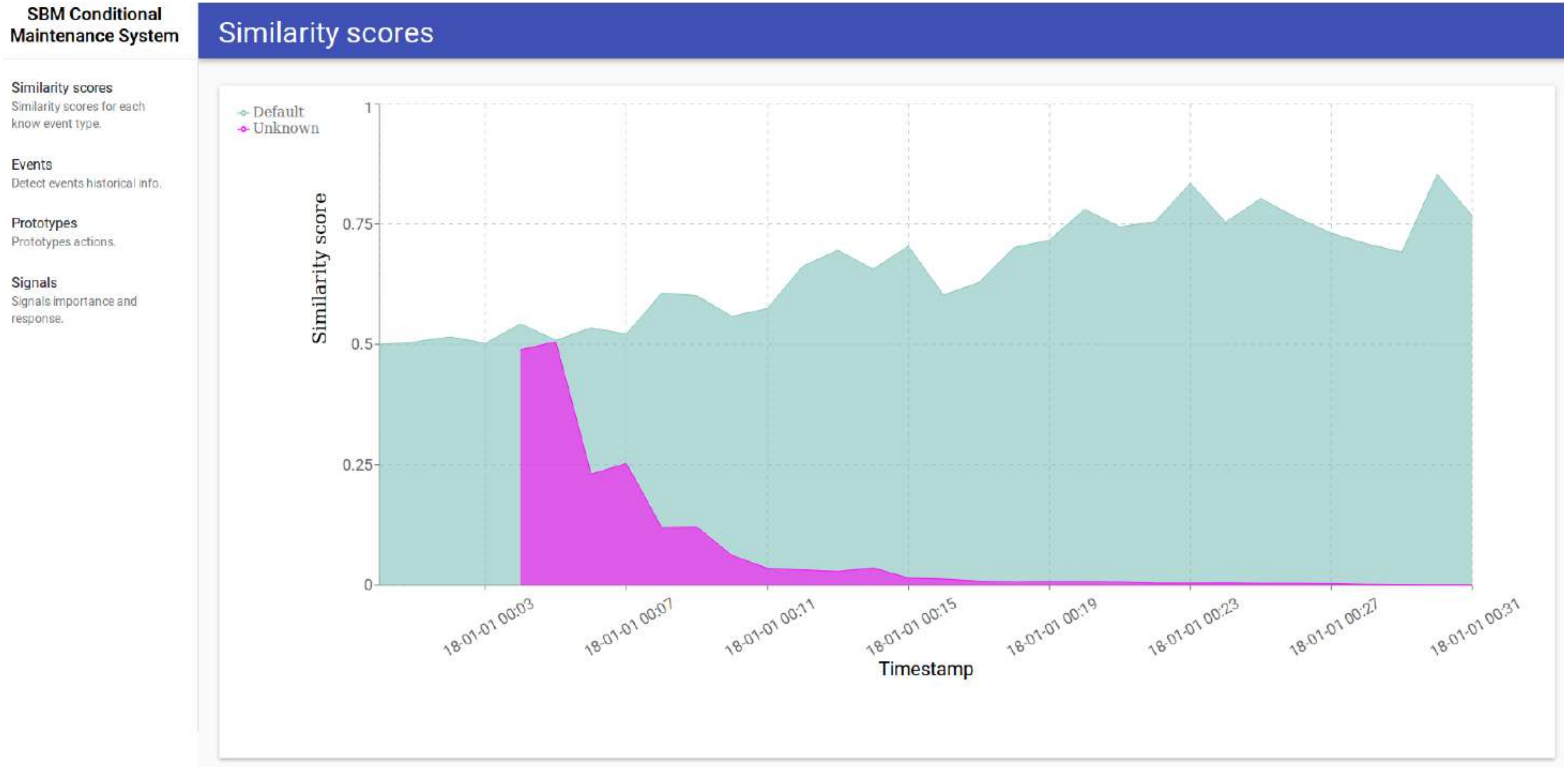
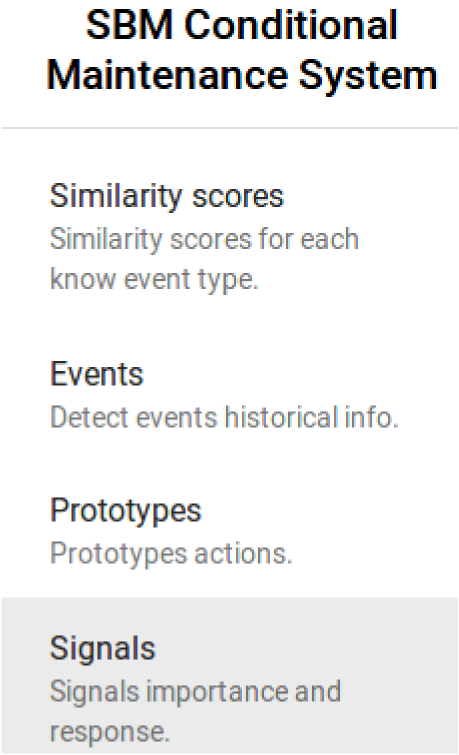


Figure A.1: Similarity score window. It presents, at each instant, each known condition score given by the detection system.

Following Table A.1, Section A.2 presents the similarity window, its components and how the similarity scores are computed. Section A.3 describes the event window, its components and functionalities, selecting and editing components. In a similar manner, the prototype window components and functionalities, selecting, visualizing, editing, and deleting prototypes, are discussed in Section A.4. Lastly, Section A.5 presents the signal window and its components.

A.2 Similarity score window

The first window of the application. As depicted in Figure A.1, it is composed of two elements: a navigation menu, at the left size of Figure A.1, and show in detail in Figure A.2; and a chart presenting the similarities scores over time for each class.



SBM Conditional Maintenance System

Similarity scores
Similarity scores for each know event type.

Events
Detect events historical info.

Prototypes
Prototypes actions.

Signals
Signals importance and response.

Figure A.2: Application navigation menu with a brief description of each window.

The navigation menu, depicted in Figure A.2, is always visible during the application execution, allowing a user to move between different windows anytime. It also shows a brief description of each window function below the window name, which serves as a small guide of the functionalities. Each entry in the menu is a link to the respective window. Thus to change between windows the user only has to click at the chosen window entry in the menu.

A.2.1 Similarity score chart

The second component of the similarity score window, the similarity score chart allows visualizing historical scores data for each known class. Depicted in Figure A.3, the similarity score chart shows, for each known class, its similarity score along time. The chart is composed of a legend, a list of known classes and their graphical representation in the chart; two axis, the similarity score axis, always between 0 and 1, and the timestamp axis, in minutes; and the scores values for each class. This chart enables the user to visualize possible changes in the equipment behavior along time.

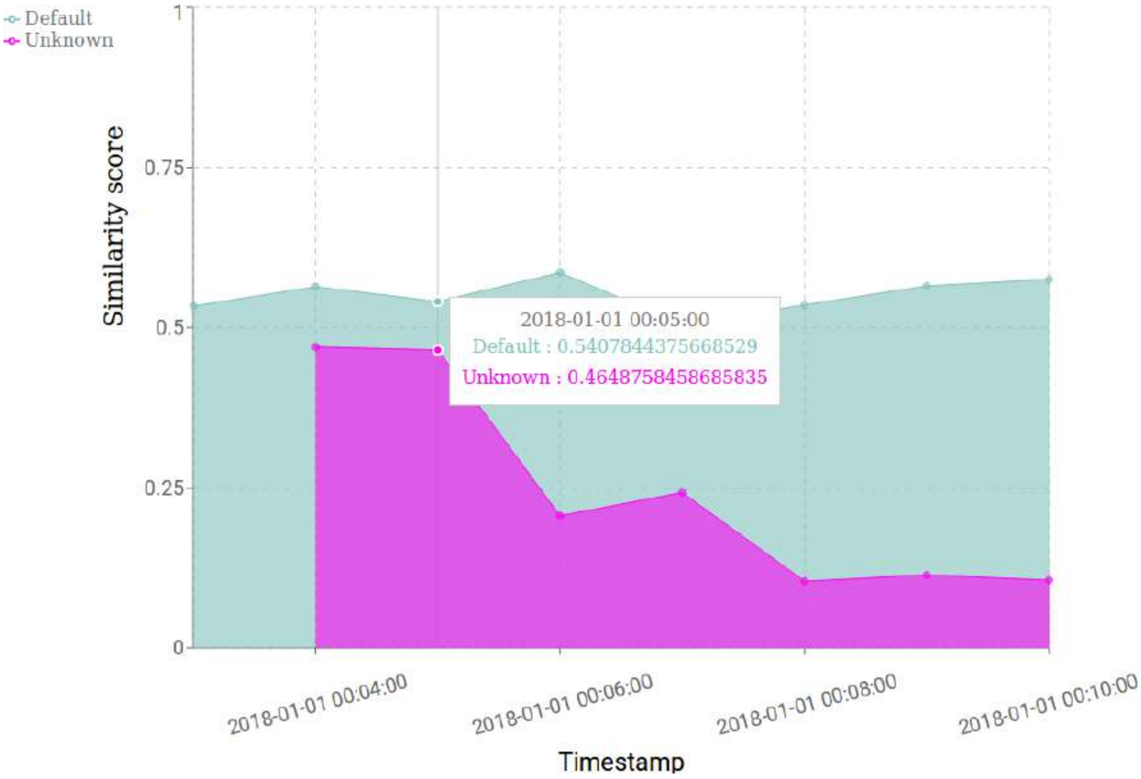


Figure A.3: Similarity score chart with each known class similarity score.

Also, as shown in Figure A.3, if the user hovers the mouse over the chart, it presents details about its data points, including the nearest data point timestamp and similarity score values.

A.2.2 Similarity score computation

The similarity scores computation follows an online learning SBM algorithm that can be divided in 5 steps:

1. *Initialization*: since the online algorithm works almost in an unsupervised

fashion, it assumes that the first sample comes from a healthy state to initialize

$$\mathcal{P}_0 \leftarrow \{\mathbf{x}_0\}\,. \tag{A.1}$$

Therefore, when this assumption fails the operator should correct the system with the correct class;

2. *Prediction error*: Given a set of prototypes $\mathcal{P}_n$ at instant n, during the prediction step the error between the current sample $\mathbf{x}_n$ and each prototype is computed as follows

$$\mathbf{e}_{np} = \Big[|x_{n1} - x_{p1}| \quad |x_{n2} - x_{p2}| \quad \ldots \quad |x_{nm} - x_{pm}|\Big]^{\mathrm{T}}, \quad p \in \mathcal{P}; \tag{A.2}$$

3. *Instance similarity*: the prediction error $\mathbf{e}_{np}$ for each prototype is used as input for the similarity function $f_n(\bullet)$, which follows a logistic regression model that computes the similarity as the probability of any two points to belong in the same set as

$$s_{npc} = \left\{1 + \mathrm{e}^{\left[d^2_{\mathbf{A}_c}(\mathbf{x}_n, \mathbf{x}_p) - \mu_c\right]}\right\}^{-1}, \tag{A.3}$$

where parameter μ_c represents the distance threshold separating if points are in the same set, parameter $\mathbf{A}_c \in \mathbb{R}^{m\times m}$ is the distance metric matrix, and $d^2_{\mathbf{A}_c}(\mathbf{x}_i, \mathbf{x}_j)$ is

$$d^2_{\mathbf{A}_c}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}} \mathbf{A_c} (\mathbf{x}_i - \mathbf{x}_j)\,. \tag{A.4}$$

4. *Predicted similarity and class*: computed as

$$g_{nc} = f_n(|\mathbf{x}_n - \hat{\mathbf{x}}_{nc}|)\,, \tag{A.5}$$

$$c^* = \arg\max_c\{g_{nc}\}. \tag{A.6}$$

The g_{nc} are the scores plotted in the similarity scores plot for each class c.

5. *Update*: Lastly, the update strategy first tries to add a new prototypes when necessary, then updates the similarity function. The prototype insertion is made using the following heuristic:

$$\mathcal{P}_n \leftarrow \mathcal{P}_{n-1} \cup \{\mathbf{x}_n\} \quad \text{if} \begin{cases} P_{n-1} < P_{\max}; \\ \wedge \quad g_n \le \tau; \\ \wedge \quad g_{n-1} > g_n. \end{cases} \tag{A.7}$$

where $P_n = |\mathcal{P}_n|$ and $P_{\max}$ are the current number of prototypes and maximum allowed number of prototypes, respectively, and $g_n = \max_c\{g_{nc}\}$. The threshold τ is the minimum similarity and in this work is empirically set as $\tau = 0.7$.

Then the similarity function $\mathbf{A}_c$ and μ_c parameters are updated by computing their gradients assuming that the chosen label c^* is the correct class, with

$$y_{npc} = \begin{cases} +1 & \text{if } c = c^*; \\ -1 & \text{otherwise.} \end{cases} \tag{A.8}$$

and

$$\mu_c^{n+1} = \mu_c^n + \sum y_{npc} \left(s_{npc} - 1\right), \tag{A.9}$$

$$\mathbf{A}_c^{n+1} = \mathbf{A}_c^n - \sum y_{npc} \left(s_{npc} - 1\right) \left(\mathbf{x}_n - \mathbf{x}_p\right) \left(\mathbf{x}_n - \mathbf{x}_p\right)^{\mathrm{T}} - \gamma 2 \mathbf{A}_c^n. \tag{A.10}$$

A.3 Event window

The event window, shown in Figure A.4, has the objective of allowing a user to select, review, and edit detected events. It is composed of an event info dialog, which presents information about the chosen event and allows its selection and edition; and the similarity scores charts, which allows visualizing how the event developed by presenting the similarity scores during the event occurrence. The next sections describes what happens when a new event is detected and possible actions of an operator in this case: selecting, review, and editing the occurrence.

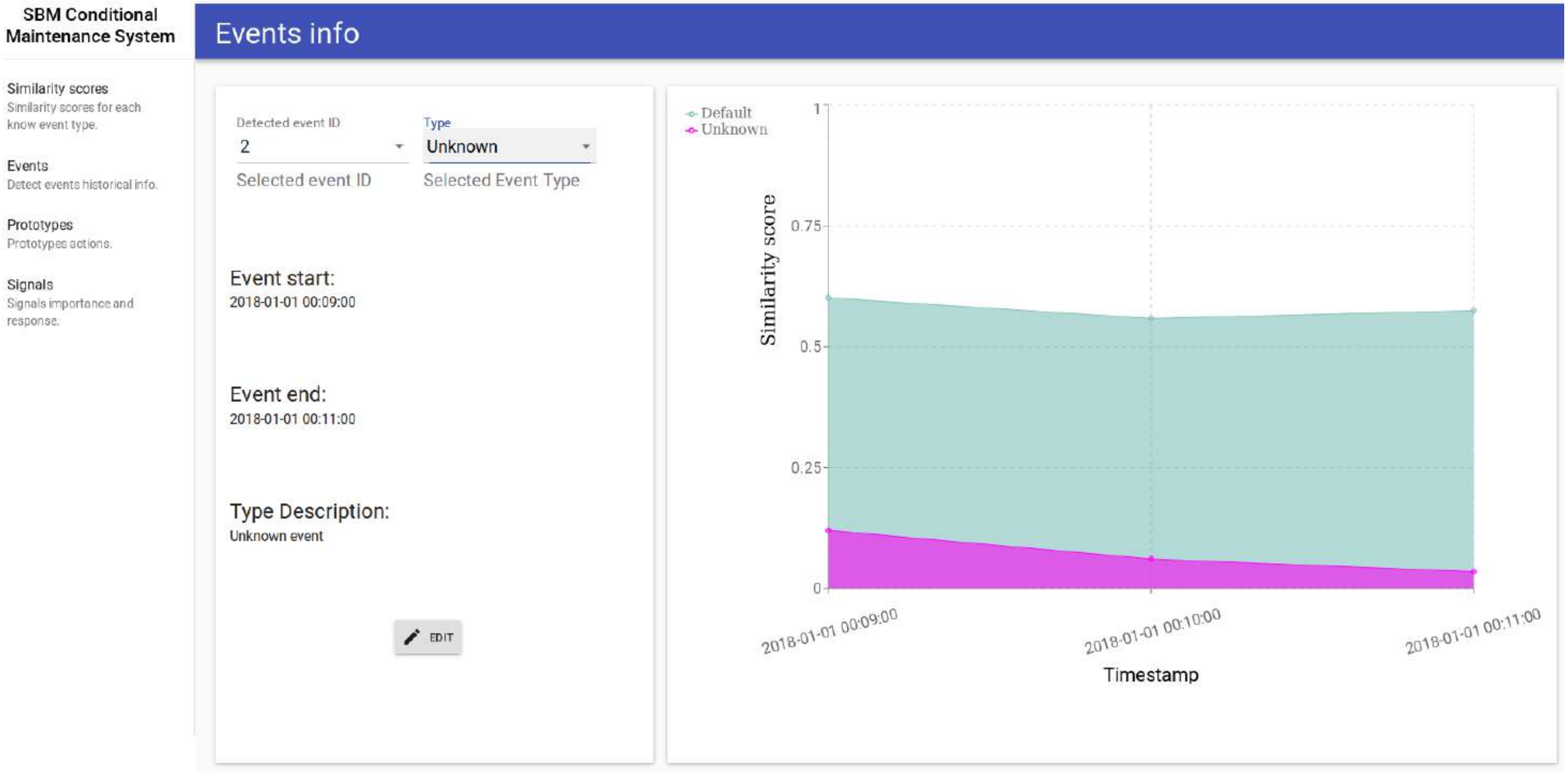
A.3.1 Event occurrence

An event occurs if:

- a new prototype is detected. In this case it is considered a new anomalous event; or
- a non-healthy event achieves the greatest score. In this case the fault is identified as the event with the greatest score.

Non-healthy events include fault, shutdown, or unknown events. Fault events are the ones identified by an operator as fault or failure states. Shutdown events are those events where the equipment was turned off or is in stand-by but the sensors and the detection system where not turned off.

Lastly, unknown events are the ones which produced a new prototype but were not reviewed by a human operator, thus were not identified as belonging to any of the other types. These events should be identified and edited when detected. otherwise,



Detected event ID: 2
Selected event ID

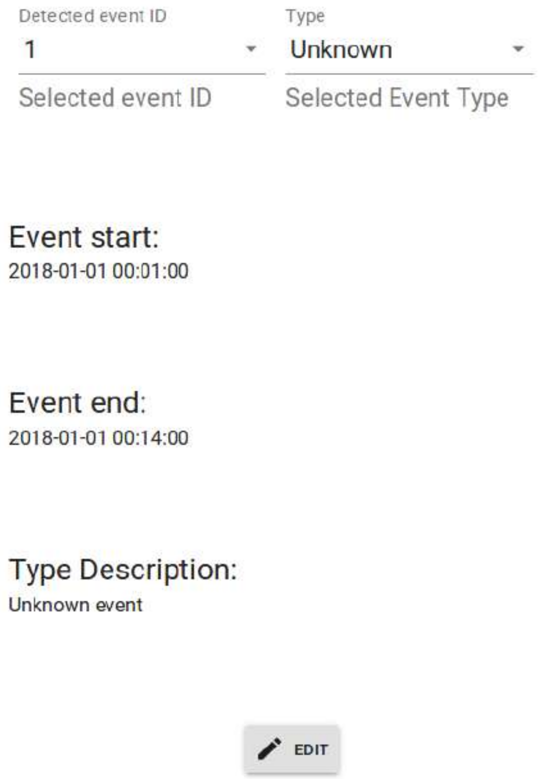
Type: Unknown
Selected Event Type

Event start:
2018-01-01 00:09:00

Event end:
2018-01-01 00:11:00

Type Description:
Unknown event

Figure A.4: Event window. It allows a user to selected a detected event, edit, and review its historical data.



Detected event ID
1
Selected event ID

Type
Unknown
Selected Event Type

Event start:
2018-01-01 00:01:00

Event end:
2018-01-01 00:14:00

Type Description:
Unknown event

Figure A.5: Events info dialog. It shows event description and allows event selection and edition.

they could produce erratic system behavior, as each following new prototype could be assigned to the same event type.

After an event is detected, it is registered with an id for future review and edition. An event can be inspected by means of the event info dialog, depicted in Figure A.5. This dialog presents the following fields:

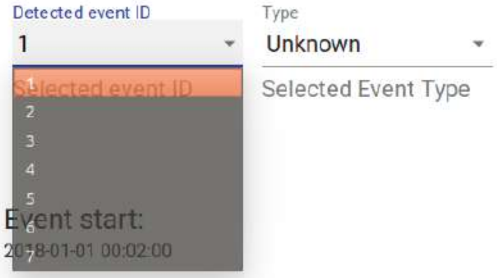
- *Event id*: current selected event id, an unique integer identifying the event. This a selection field that can be used to chose between any detected event;

- *Event type*: selected event type. This field is not selectable, but changes with the selected event. Only four options are allowed in this field: *healthy*, representing a healthy state; *shutdown*, representing any state where the equipment was not working; *faulty*, representing a faulty equipment condition; and *unknown*, where the condition needs to be identified by an operator.

- *Event start*: timestamp where the selected event occurrence was first detected by the system;

- *Event end*: timestamp where the selected event stopped being detected by the system;

- *Type description*: the event type description.



Detected event ID
1
Selected event ID

Type
Unknown
Selected Event Type

Event start:
2018-01-01 00:02:00

Figure A.6: Event selection. Current selected event has id 1.

At the bottom of the event dialog is the *edit* button which opens the event type edition dialog, allowing a user to change the selected event type to other known type. This action is discussed at the next section.

A.3.2 Selecting an event

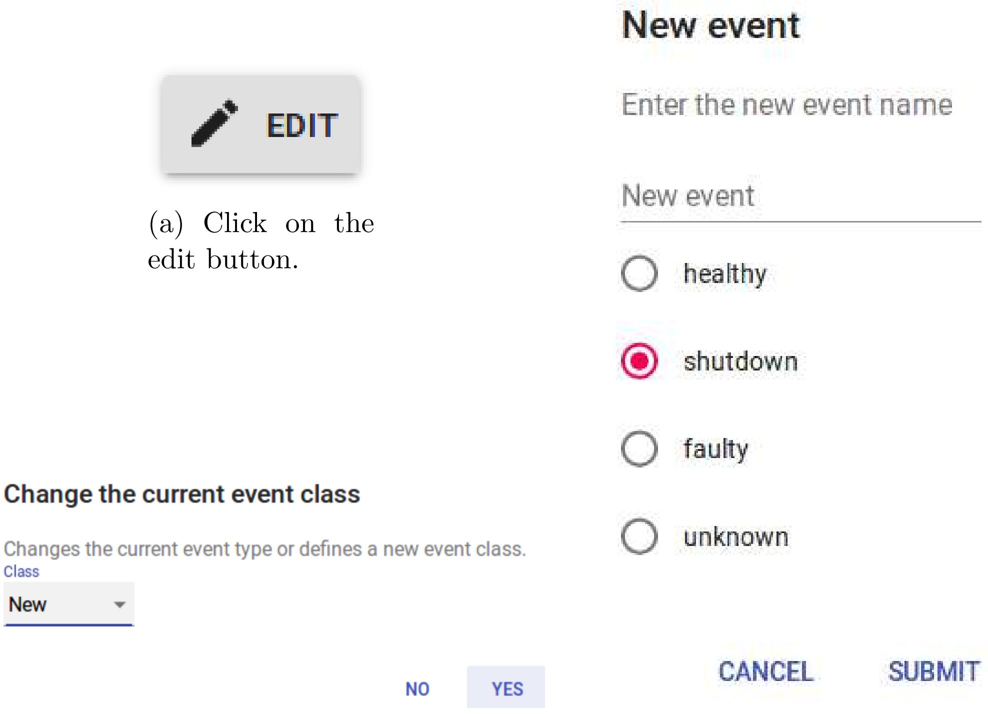
To select an event just click on the current event id number. This action opens a drop-down menu presenting all the current detected events represented by their id numbers, as shown in Figure A.6. The user should either click on the desired id number or move between the numbers using the keyboard arrows (↑ to go up and ↓ to go down).

A.3.3 Editing an event

The process of editing an event is depicted in Figure A.7. First, the user must select the event which should be edited by following the selection procedure shown in Figure A.6.

After that, the user should click on the *edit* button (Figure A.7a). This action opens the type edition dialog, illustrated by Figure A.7b. There the user can select any previously known state or select the *New* state and confirm or cancel this action by clicking in *Yes* or *No*, respectively. If the selection is confirmed and a previously known event state is selected, then no further action is required and the event is edited to the new event type.

However, if the *New* event option is selected, then a new dialog, depicted in Figure A.7c opens. This dialog has a text input field where the user can write the new event name and radio buttons with the four event types. After writing the event name and the type, the user can either confirm this edition (clicking on *Yes*) or cancel the action (clicking on *No*). If the action is confirmed, the new event class is submitted and the event is edited to this class.



Change the current event class

Changes the current event type or defines a new event class.

Class: New

New event

Enter the new event name

New event

○ healthy

● shutdown

○ faulty

○ unknown

(a) Click on the edit button.

(b) Select the correct event class.

(c) If *New* event is selected, one most submit the new event name and type.

Figure A.7: Event edition procedure. Each step is detailed in text.

A.4 Prototype window

The prototype window enables the user to review, visualize, edit or even delete a *prototype*. A prototype is a representative sample from the observed samples stored by the system given its quality into assisting detecting similar samples. Since the prototype occurred during a given state of the equipment, similar samples are assumed to belong to the same condition.

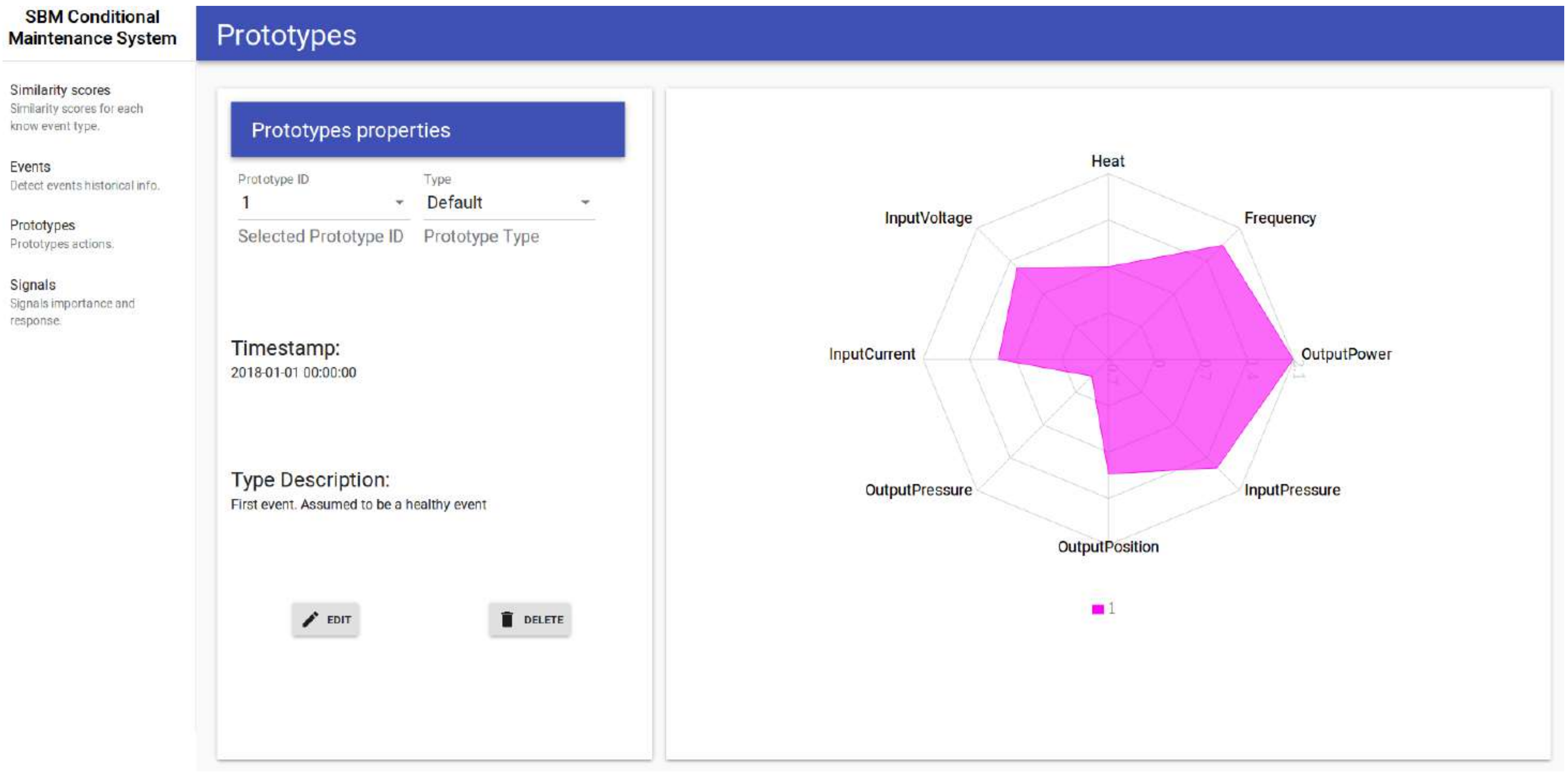
Hence, the system has at least one prototype for each known condition. Since the prototypes are representative of the condition, an experienced operator could use information from the prototype to correct the system decision, to assert the current condition, and to delete wrong prototypes. The prototype windows, depicted in Figure A.8, facilitates these tasks.

As with the event window, the prototype window is composed of two elements: the prototype description, which presents information about a chosen prototype; and a radar chat of the prototype attributes.

A.4.1 New prototype detection

A new prototype is detected when:

- if the maximum number of prototypes was not reached;



SBM Conditional Maintenance System

Prototypes

Similarity scores
Similarity scores for each know event type.

Events
Detect events historical info.

Prototypes
Prototypes actions.

Signals
Signals importance and response.

Prototypes properties

Prototype ID: 1
Selected Prototype ID

Type: Default
Prototype Type

Timestamp:
2018-01-01 00:00:00

Type Description:
First event. Assumed to be a healthy event

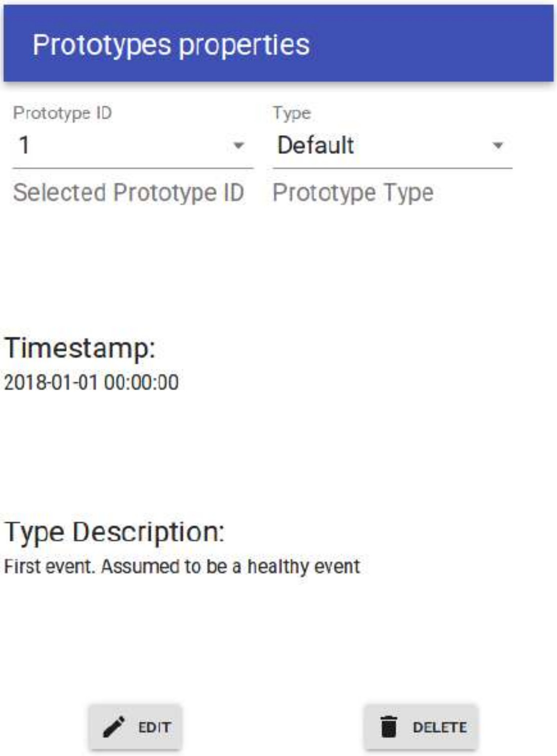
Figure A.8: Prototype window. It allows to select a prototype for visualization, displaying relevant information, and for edition or deletion.

- the maximum similarity between all current classes is below a given threshold;

- and its smaller than the previous maximum similarity, indicating a consistent reduction in similarity.

If all these conditions are satisfied, then it is assumed that the current state is not being correctly represented by any of the prototypes and should be its own representative.

Since this occurrence also means that no event is sufficiently similar to the current condition of the equipment, a new event is triggered and labeled as *Unknown*. In case of a false positive, the operator should ignore this event and delete the prototype. Otherwise, the operator should edit the event and the prototype to their correct condition.



Prototypes properties

Prototype ID
1
Selected Prototype ID

Type
Default
Prototype Type

Timestamp:
2018-01-01 00:00:00

Type Description:
First event. Assumed to be a healthy event

Figure A.9: Prototypes description dialog. It shows prototype description and allows prototype selection, edition and deletion.

A detected prototype can be reviewed, edited, and deleted using the prototype description dialog, shown in Figure A.9. This dialog presents the following fields:

- *Prototype id*: current selected prototype unique integer id. Can be used to select a prototype;

- *Prototype event type*: Event type related with the selected prototype. Normally the condition that occurred when the prototype was identified;

- *Detection timestamp*: Moment when the selected prototype was detected;
- *Prototype event description*: Description of the event associated with the selected prototype.

At the bottom of this dialog there are two buttons: the *edit* button, which allows editing the event related with the selected prototype; and the *deletion* button, which allows deleting a prototype, mostly likely a false positive. The procedures related with each button are described in the next sections.

A.4.2 Selecting a prototype

This procedure is very similar the event selection procedure. Just click on the current prototype id number. This action opens a drop-down menu with all prototype ids, as illustrated by Figure A.10. The user should either click on the desired id number or move between the numbers using the keyboard arrows (↑ to go up and ↓ to go down).

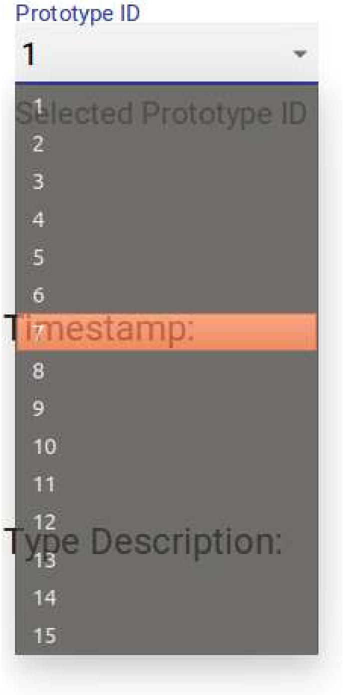
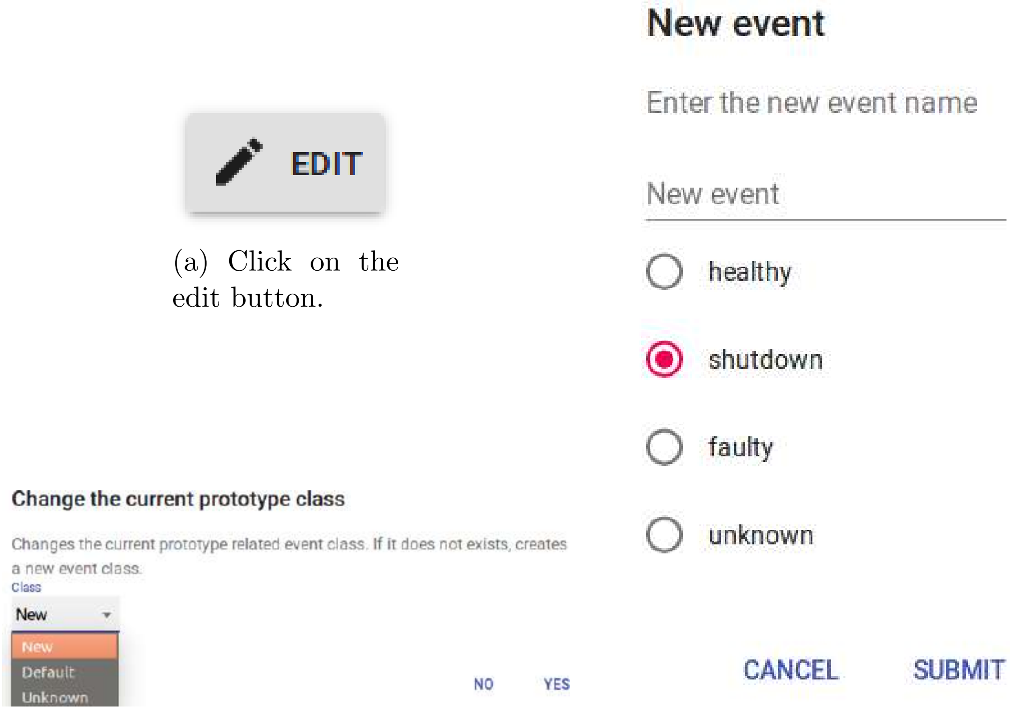


Figure A.10: Prototype selection. Current selected event has id 7.

A.4.3 Editing a prototype

Editing a prototype follows almost the same logic of editing an event, with each step described in Figure A.11. First, the user selects the target prototype. Then, the user should click on the *edit* button (Figure A.11a). After this step the user can select the correct event class using the drop-down menu, as shown in Figure A.11b. After choosing between any of the known event classes or *New*, the user can either confirm the decision by clicking on *Yes*, or canceling the edition by clicking in *No*. If the edition is confirmed and *New* is not selected, no further action is required. Otherwise, the new event dialog opens, as shown in Figure A.11c.

This new event dialog requires the same actions as it counterpart presented in the event edition procedure and produces the same results. It has a text input field for the new event name and radio buttons with the event type to be selected. After the form is completely filled, the user can either confirm this new event by clicking on *Yes* or cancel the action by clicking on *No*. If the edition is confirmed, the new event class is submitted and the selected prototype becomes associated with this new class.



(a) Click on the edit button.

Change the current prototype class

Changes the current prototype related event class. If it does not exists, creates a new event class.

Class: New

(b) Select the correct prototype class.

New event

Enter the new event name

New event

○ healthy

● shutdown

○ faulty

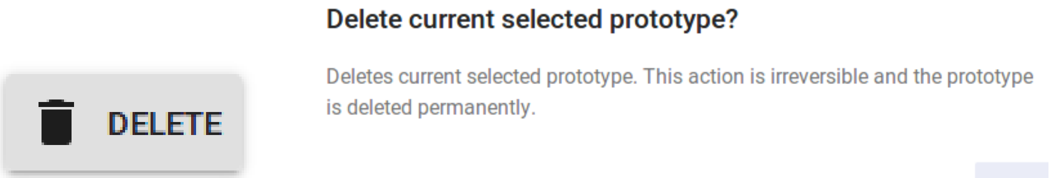
○ unknown

(c) If *New* event is selected, one most submit the new event name and type.

Figure A.11: Prototype edition procedure. Each step is detailed in text.

A.4.4 Removing an prototype

The steps to delete a prototype are shown in Figure A.12. This procedure is very straightforward. First the user clicks on the *delete* button (Figure A.12a). This opens the deletion dialog (Figure A.12b), warning the user about the consequences of a deletion, which is an irreversible act, and asking for confirmation. If the user clicks on *Yes* the prototype is permanent deleted. Otherwise, the action is canceled and nothing happens.



Delete current selected prototype?

Deletes current selected prototype. This action is irreversible and the prototype is deleted permanently.

(a) Click on the delete button.

(b) Confirm or cancel the deletion.

Figure A.12: Deletion procedure. This procedure is irreversible and should be made with caution.

A.4.5 Prototype radar chart

The *prototype radar chart*, depicted in Figure A.13, presents each attribute of a single prototype at the same chart, allowing a fast visualization, leading to a fast identification of problems, anomalous characteristics, and incorrectly identified prototype. Extra information about each attribute can be visualized by hovering the mouse over the chart.

A.5 Signal window

The application's fourth and last window, this window enables the user to review the registered signals and to visualized the measured signal importance. In the future, it should have options to add, remove, or edit registered signals or their descriptions. However, these functionalities were only implemented in the backend and will be implemented in the future. Depicted in Figure A.14, it is composed of a table of registered signals and the signal importance pie chart.

A.5.1 Registered signal table

The registered signals table, shown in Figure A.15, presents the list of current registered signals, including their internal id, name, SI unit, and description. It allows selecting the size of the viewable list without between 5, 10 and 25 rows using the selector presented in Figure A.16. The remaining signals can be viewed by paginating the list using the page selector show in Figure A.17. This component shows the total number of signals and allows moving between pages. In the future, these component should also allow sorting and selecting components from the table.

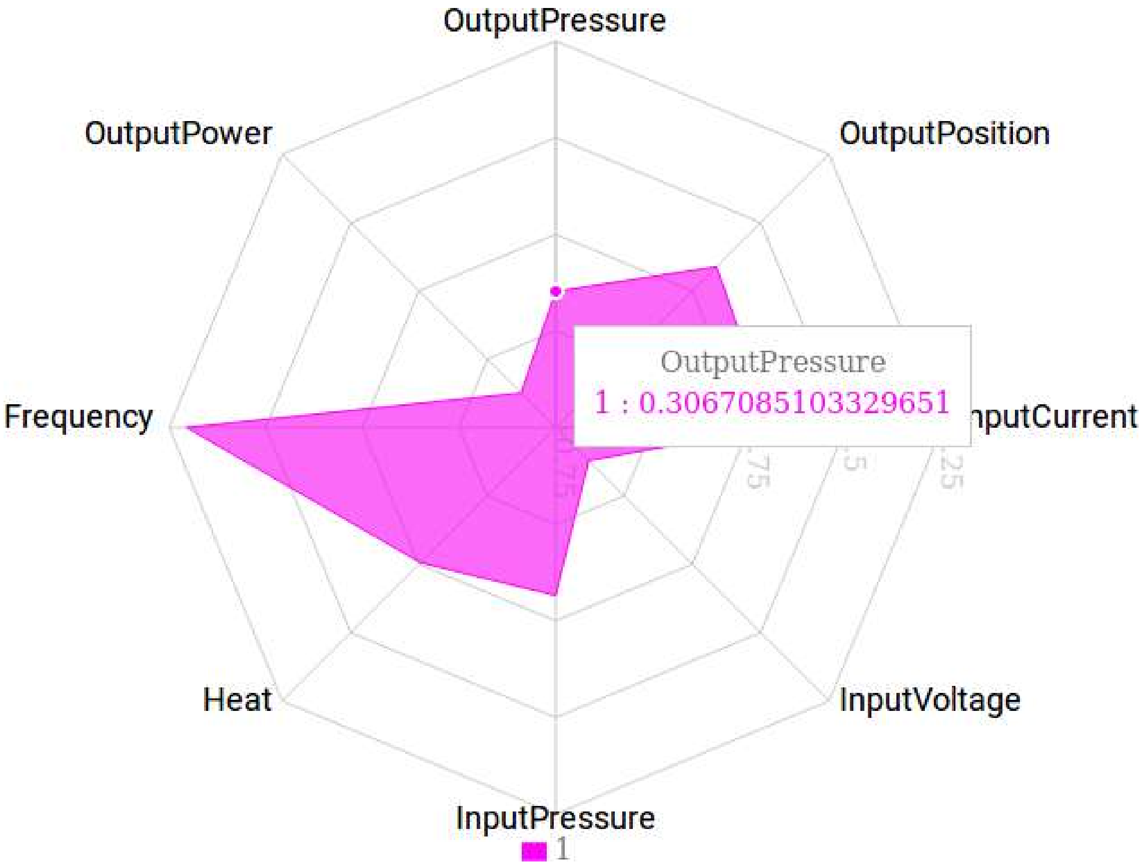


Figure A.13: Radar chart with mouse over the *OutputPressure* point.

A.5.2 Signals importance chart

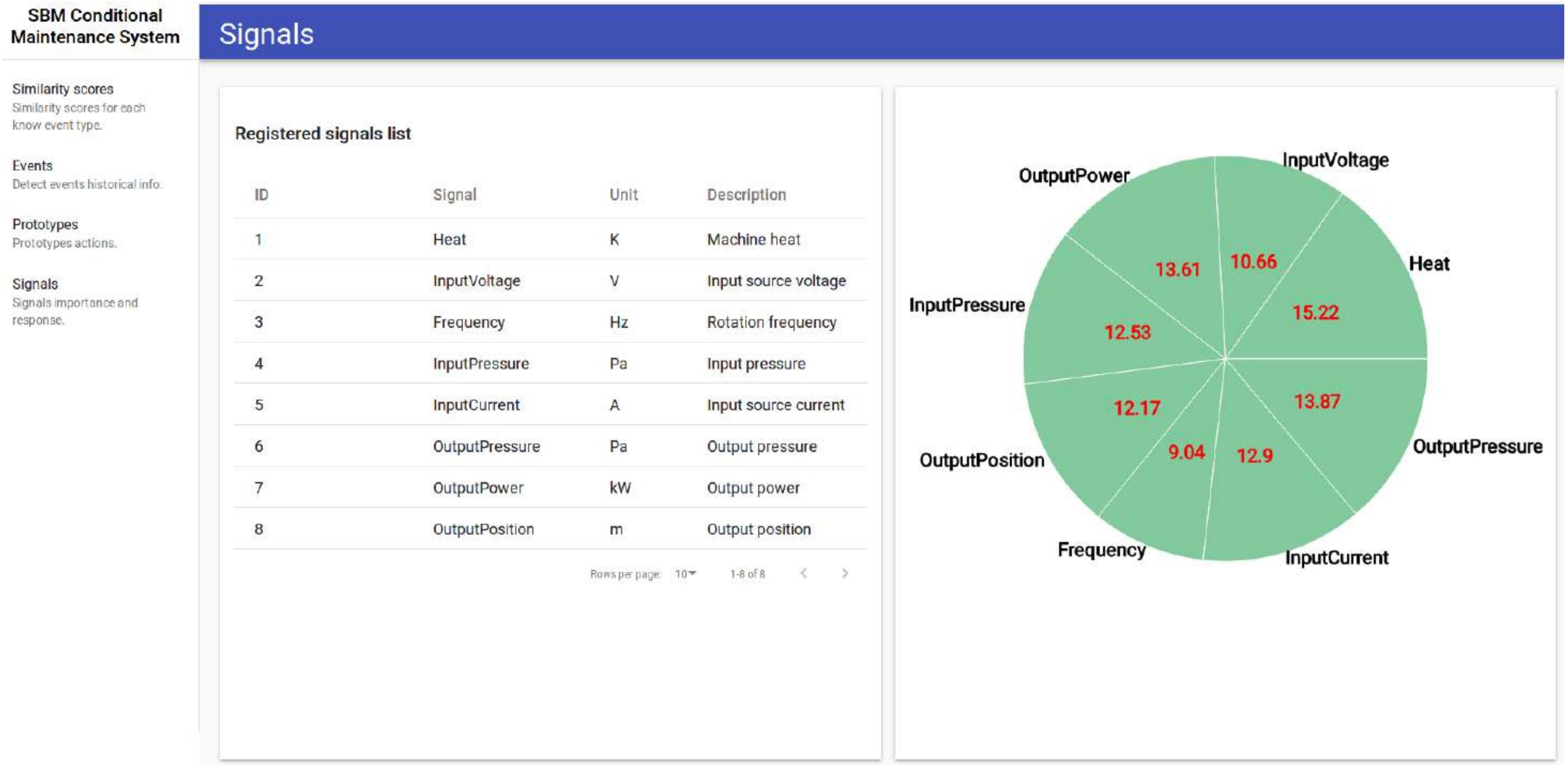
The signals import pie chart, shown in Figure A.18, is the represents the current relative importance of each signal in the current detection, making it useful for detecting possible problems in the sensors, as a defective sensor would either gain a higher importance or become irrelevant.

The importance value are computed using an approach similar to the *permutation importance* used in random forests [59]: given the current sample $\mathbf{x}_n$, we produce, for each prototype $\mathbf{x}_p$ and error $\mathbf{e}_{np}$, a perturbed version where a single measurement at m is permuted by its original value with random Gaussian noise σ_m, producing a new error

$$\hat{\mathbf{e}}_{np}^{m} = \mathbf{e}_{np} + \mathbb{1}_m \sigma_m, \tag{A.11}$$

where $\mathbb{1}_m$ denotes the vector with a 1 in the m-th coordinate and 0's elsewhere. Then, for each measurement m we can compute the deviation between the original g_{nc} and the perturbed g_{nc}^{m} as

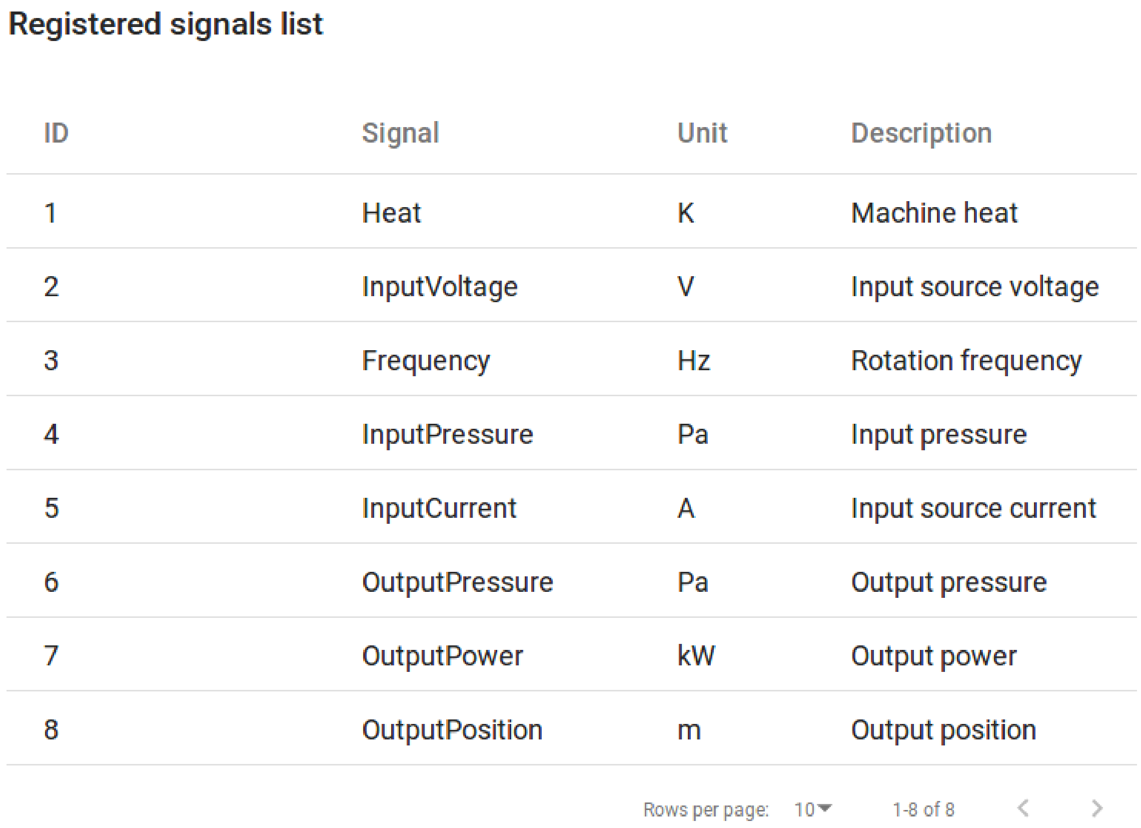
$$\widehat{\text{imp}}_{mc} = \left| g_{nc} - g_{nc}^{m} \right|, \tag{A.12}$$



Registered signals list

ID	Signal	Unit	Description
1	Heat	K	Machine heat
2	InputVoltage	V	Input source voltage
3	Frequency	Hz	Rotation frequency
4	InputPressure	Pa	Input pressure
5	InputCurrent	A	Input source current
6	OutputPressure	Pa	Output pressure
7	OutputPower	kW	Output power
8	OutputPosition	m	Output position

Figure A.14: Signal window. This windows shows the list of registered signals and their current relative importance as a pie chart.



Registered signals list

ID	Signal	Unit	Description
1	Heat	K	Machine heat
2	InputVoltage	V	Input source voltage
3	Frequency	Hz	Rotation frequency
4	InputPressure	Pa	Input pressure
5	InputCurrent	A	Input source current
6	OutputPressure	Pa	Output pressure
7	OutputPower	kW	Output power
8	OutputPosition	m	Output position

Figure A.15: Registered signal table. It presents a list of the current registered signals.

Rows per page: 10▾

Figure A.16: Table number of rows selector. It allows selecting between 5, 10 and 25 signals per page.

1-8 of 8 < >

Figure A.17: Page selector with number of signals in current page. Can be used to visualize all the list by paginating.

and the signal relative importance as

$$\mathrm{imp}_{mc} = \frac{\widehat{\mathrm{imp}}_{mc}}{\sum_{kc} \widehat{\mathrm{imp}}_{kc}}. \tag{A.13}$$

Since the sum of relative importances is equal to 1, we can convert each signal importance into a percentage by multiplying each imp_{mc} by 100 before plotting in the signal importance pie chart (Figure A.18). Only the importance give the current detected class is plotted.

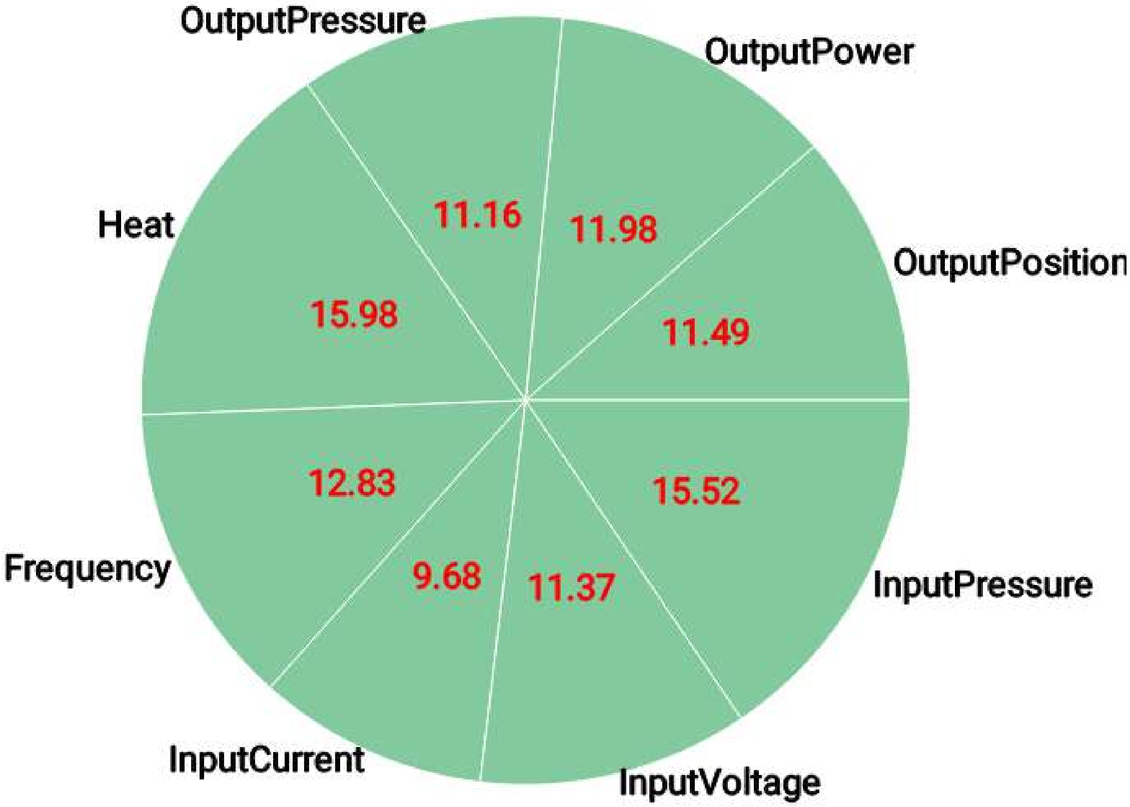


Figure A.18: Signal importance pie chart. It shows the relative importance of each registered signal given the equipment current condition.