HALMSTAD
UNIVERSITY

Hierarchical Methods for Self-Monitoring Systems: Theory and Application

Pablo José Del Moral Pastor

Supervisors:
Slawomir Nowaczyk
Sepideh Pashami

DOCTORAL THESIS | Halmstad University Dissertations no. 93

Hierarchical Methods for Self-Monitoring Systems: Theory and Application

Halmstad University Dissertations no. 93
ISBN 978-91-88749-98-7 (printed)
ISBN 978-91-88749-97-0 (pdf)
Publisher: Halmstad University Press, 2022 | www.hh.se/hup
Printer: Media-Tryck, Lund

Abstract

Self-monitoring solutions first appeared to avoid catastrophic breakdowns in safety-critical mechanisms. The design behind these solutions relied heavily on the physical knowledge of the mechanism and its fault. They usually involved installing specialized sensors to monitor the state of the mechanism and statistical modeling of the recorded data. Mainly, these solutions focused on specific components of a machine and rarely considered more than one type of fault.

In our work, on the other hand, we focus on self-monitoring of complex machines, systems composed of multiple components performing heterogeneous tasks and interacting with each other: systems with many possible faults. Today, the data available to monitor these machines is vast but usually lacks the design and specificity to monitor each possible fault in the system accurately. Some faults will show distinctive symptoms in the data; some faults will not; more interestingly, there will be groups of faults with common symptoms in the recorded data.

The thesis in this manuscript is that we can exploit the similarities between faults to train machine learning models that can significantly improve the performance of self-monitoring solutions for complex systems that overlook these similarities. We choose to encode these similarity relationships into hierarchies of faults, which we use to train hierarchical supervised models. We use both real-life problems and standard benchmarks to prove the adequacy of our approach on tasks like fault diagnosis and fault prediction.

We also demonstrate that models trained on different hierarchies result in significantly different performances. We analyze what makes a good hierarchy and what are the best practices to develop methods to extract hierarchies of classes from the data. We advance the state-of-the-art by defining the concept of heterogeneity of decision boundaries and studying how it affects the performance of different class decompositions.

Al bicho y a la madre del bicho.

Acknowledgments

One could wrongly assume that a PhD is the result of individual work and merit. There has been a fair amount of individual work, I can promise; but there has been extensive, invaluable and instrumental support and contribution from many others.

First and foremost, I would like to thank my principal supervisor Sławomir Nowaczyk. From the beginning of this journey, you have guided me, challenged me and helped me. You have had the wisdom to allow me to get lost in my own rabbit holes and pull me out of them when it was not too late. If I have an understanding of what scientific research is, and what it means to be a researcher, it is thanks to you.

I want to also thank Sepideh Pashami, Anita Sant'Anna and Stefan Byttner for their roles as co-supervisors. Sepideh, even though you were my friend before my supervisor, you were able to contribute to my work critically and supportively. Anita, you were great at identifying my greatest weaknesses and challenging me to improve them, and for that I am thankful. Stefan, first as co-supervisor and then as a lab leader, you have always had a helping hand; I never left your office with a doubt or a problem unsolved.

The work leading to this thesis has been carried out in parallel and collaboratively with the PRIME project. I would like to thank Getinge AB in general, and the people I have collaborated specifically. Jean-Pierre, thanks for your support and friendship. Mikael, Rasmus, Karsten, William, Soren, it was always a pleasure to meet you on Friday mornings, thanks for all your support and invaluable insights.

I would like to thank my parents Sigfrido and Carmen for the continuous support during my life, this thesis is also the result of your efforts. I would also like to thank my parents in love Ali and Nilufer, for their great support and help during the different deadlines regarding this thesis.

I should not write acknowledgements without mentioning all the great friends that underwent or are still undergoing the pleasures of PhD life: Deicy, Kevin, Maytheewat, Yuantao, Hassan, Sebastian, Kunru, Suleyman, Saeed, Awais, Jennifer, Anna, Tiago, Zahra, Mahsa, Abdallah, Ghaith, Nasrin... sharing the experiences and complaints of PhD lives has been critical in maintaining some level of mental health. I want to also thank all the colleagues in the lab: Tommy, Eric, Thomas, Marlena, Denni, Björn, Josef, Fernando, Peyman, Mahmout, Farzaneh, Antanas, Roland, Mattias,... I am probably forgetting someone important, but let's be honest, acknowledgements are the last thing one writes, and I am already late for printing.

But not too late to remember my loving wife. Ececita, what a journey! The day you first crossed the doors of the lab and how you conveniently lost the remote of the TV are the two defining moments of my life that shaped a happiness that I did not was possible. Doing two PhDs in a household, while building a family and keeping the electricity costs at bay has been a challenge; but I would do it again and again. I owe everything to you and your love. I should also mention Ada. At two years of age, you have not contributed much to this thesis (actually I owe to you some delays and missing meetings), but everything we are doing is for you.

¡Muchas gracias!

Contents

List of Figures

List of Tables

List of Algorithms

List of Papers

This thesis summarizes the following papers.

(I) Pablo del Moral, Sławomir Nowaczyk, Sepideh Pashami. Filtering Misleading Repair Log Labels to Improve Predictive Maintenance Models. *PHM Society Conference*, 7(1), pp. 110-117, 2022

(II) Pablo del Moral, Sławomir Nowaczyk, Sepideh Pashami. Hierarchical Multi-class Classification for Fault Diagnosis. *Proceedings of the 31st European Safety and Reliability Conference (ESREL 2021)*, pp. 2457–2464, 2021

(III) Pablo del Moral, Sławomir Nowaczyk, Sepideh Pashami. Hierarchical multi-fault prognostics for complex systems *Expert Systems with Applications (submitted)*

(IV) Pablo del Moral, Sławomir Nowaczyk, Anita Sant'Anna, Sepideh Pashami. Pitfalls of Assessing Extracted Hierarchies for Multi-Class Classification *Pattern Recognition*, *(minor revision)*

(V) Pablo del Moral, Sławomir Nowaczyk, Sepideh Pashami. Why is Multi-Class Classification Hard? *IEEE Access*, 10, pp. 80448-80462.

Chapter 1

Introduction

Any machine inevitably degrades with usage. Degradation translates into a decrease in performance and, potentially, a breakdown that will suddenly stop the machine. Maintenance operations extend the operational lifetime of a machine, increasing its performance and avoiding potentially catastrophic breakdowns.

Every maintenance operation comes at a cost. In addition to hiring a professional, we need to factor in the time it takes for the maintenance operation and the possible price of new components to replace faulty ones.

Traditionally, there have been two main maintenance strategies: corrective and preventive.

Corrective maintenance consists of repairing a machine when a failure happens. This is, for example, the typical maintenance strategy used with most household machines. Its main advantage is minimizing maintenance costs; only broken parts are replaced and only when they have exhausted their operational life. However, there are drawbacks: performance reduction while the machine's components are degrading but have not broken yet, downtime needed to diagnose and identify the problem that caused the breakdown, non-availability of the pieces, and personnel needed to perform the maintenance.

Preventive maintenance consists of replacing components of machines based on their operation time, not on their current state. This strategy is particularly adequate for highly critical machines, such as airplanes, or highly critical parts like a car's brakes. In these cases, components are replaced before there is any risk of breakdown. The drawback to this approach is that we are potentially replacing components or machines that are not yet defective, increasing overall operational costs.

Predictive maintenance is a newer maintenance strategy that aims to keep the benefits of preventive maintenance while minimizing maintenance costs. With the appropriate tools and knowledge, we can evaluate the current state of the machine. If the early symptoms of a fault are detected, the corresponding maintenance operation can be scheduled and performed before any real risk of failure. If no defect is detected, the machine does not need to be maintained and continues its normal operation. Predictive maintenance is also known as condition-based maintenance since we are making maintenance decisions based on the current state of the machine.

An example of predictive maintenance in our daily lives is the annual inspection of cars, where a technician manually checks the status of different parts. This traditional manual condition monitoring becomes extremely costly if we intend to monitor many parts on a large number of machines and with a higher frequency.

Self-monitoring solutions aim to solve this problem by automatically monitoring the condition of the machine. This is achieved through installing sensors to monitor its state, building automated solutions that interpret these sensor signals and warn of a future problem, and planning the corresponding maintenance operations.

In this thesis, we will focus on self-monitoring applications for complex machines with many different components and types of possible faults. In particular, we aim to build machine learning models to identify symptoms of ongoing faults in recorded data. Then, we use these symptoms to identify what particular fault is happening in the machine and when this fault will result in a breakdown.

1.1 Ideal scenario for self-monitoring

Let us start by defining some terms used in the thesis. Figure 1.1 illustrates a simplified and idealized version of the monitoring of a fault happening in a machine.

We can assume that the machine starts in a healthy state. The machine is being monitored with the help of some sensors or diagnostic tools that can accurately tell us the machine's state. At some moment in time, a fault happens in the machine. The fault evolves until the degradation leads to a failure/breakdown, which means the end of the operation of the machine. Once the machine has ceased its operation, a maintenance operation needs to be scheduled and performed; during this period, the machine is not operating. After the maintenance has been carried out, the machine is back to being healthy. Due to the monitoring devices installed, we can observe and measure the evolution of the fault, from the moment it starts until it fails.

The essence of a self-monitoring system is to learn to recognize the faulty patterns in the monitoring data and identify these patterns on future occasions. This valuable information can be used to improve the fault-diagnosis process and reduce the machine's downtime. It can even be used to predict a future

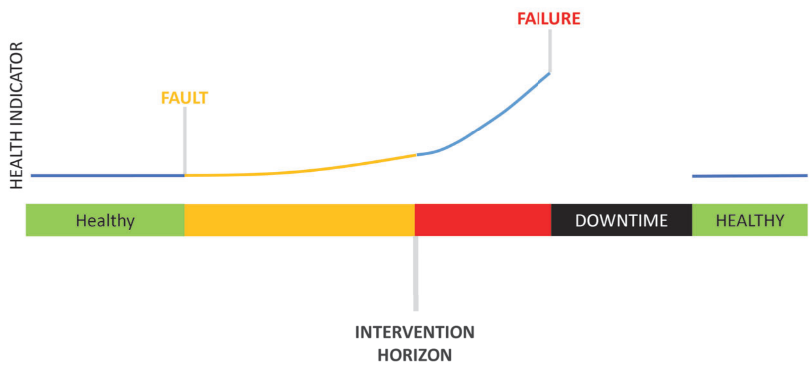


Figure 1.1: An idealized overview of the setup to train a self-monitoring system to identify faults.

failure and allow us to prevent it, averting other secondary problems that may happen as a result. A successful prediction would allow for planning a maintenance action that minimizes the machine's downtime by, for example, planning it during a natural downtime of the machine.

Planning, scheduling, and executing a maintenance operation requires time. This time needs to be accounted for when predicting afailure. A successful predictive system should be able to predict a failure with enough margin to execute the maintenance operation; we define this time as the intervention horizon.

Ideal self-monitoring system In the best-case scenario, we would want to develop a self-monitoring system during the design phase of a new machine. We could achieve this by identifying all possible faults the machine can undergo and designing monitoring devices for each component that accurately reflects its state. Then, we should design tests in controlled lab conditions to observe the different failures that can happen in the machine, detect when the fault starts and record the monitoring data describing the evolution of the degradation of the machine, together with the patterns associated with a healthy machine. Finally, machine learning models could be trained to predict and locate future failures.

In the scientific literature, most contributions to the self-monitoring systems' state-of-the-art deal with this ideal situation in different sub-problems such as fault detection, fault diagnosis, or fault prognosis. A ubiquitous example is the study of rolling bearing mechanisms [4], very dependable parts of machines. Research focuses on what signals to record [2], how to process the signals and extract useful features from them [10], and learning how to detect the faults [12], predict the failures [20], and identify when they will happen [9].

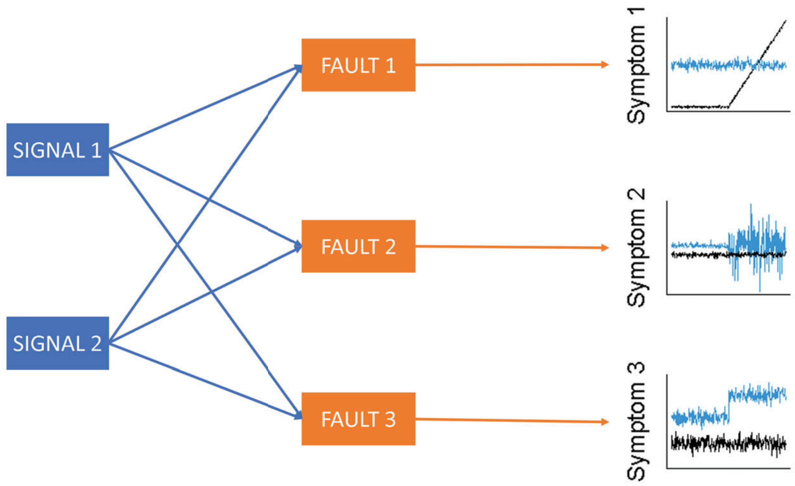


Figure 1.2: Idealized scenario for a self-monitoring system for a complex machine: every type of fault is monitored by a collection of signals; when the fault happens, a clearly distinctive symptom pattern can be observed in the data.

Rolling bearing mechanisms are relatively simple systems: typically, there are one or very few fault types. The requirements become naturally more complex if we want to design a self-monitoring solution for more complex machines. In Figure 1.2, we have a description of the ideal situation to develop a successful self-monitoring system for a complex machine. In order to monitor complex equipment with different types of potential faults, we will probably need to record more than one signal to accurately monitor the state of the machine. The main requirement in this situation is that every fault produces a distinctive pattern in the data.

Self-monitoring systems are already present in many everyday types of machinery. For example, there are sensors mounted on the wheels of a car measuring the pressure; whenever the pressure drops, an alarm is prompted for the driver to take action. In more advanced setups, the car can even schedule a visit to a selected workshop in a given margin of time to correct the fault and avert a potentially catastrophic failure.

There are deviations from this overly simplified version of the ideal self-monitoring setup, which do not break the overall framework. For example, more than one signal might be needed to monitor a given fault better, or the same fault can have more than one distinctive symptom. Other considerations can be assessed regarding the definition of what a fault is; for example, the same component can have different types of faults.

1.2 Industrial Scenario

This thesis has been carried out in the framework of the PRIME (PRedictive Intelligent Maintenance Enabler) project. The project aimed to develop a predictive maintenance system for sterilizers. This section gives an overview of the characteristics of a real-life application of self-monitoring systems, how it deviates from the aforementioned ideal scenario, and argues that some of its characteristics are common to many industrial cases.

The goal is to do self-monitoring on complex machines, understanding complexity in its many different definitions: complexity in the tasks that are carried out by the machine (a car needs to move from point A to point B, but it also needs to cool down the engine, charge the battery or cool down the cabin); complexity in the technical solutions needed to solve those tasks (each of the functional components of a sterilizer are themselves complex mechanisms managed by a complex control system that governs the global operation of the machine); and more importantly for our interest, the complexity in the number of different components that form the machine, and its various types of faults.

Challenge I: pinpointing failures during real-life operations

When we described the ideal scenario to develop a self-monitoring system, we mentioned the importance of knowing when a fault starts happening in a machine and how it was ideal for replicating these faults in lab conditions. This approach is not feasible in a real-life application of a complex machine.

In general, it will not be economical to develop lab-condition tests to monitor every single type of fault. Furthermore, there can be significant differences between how a machine is operated in a laboratory and how a machine is used in the field. External factors such as weather or environmental conditions, the load of usage, or the effect of the human factor in how the machine is used can render the lab-condition learned models useless.

In addition, complex machines such as trucks, cars, or sterilizers are expected to have a long life of operation. The design of ad-hoc self-monitoring systems in already deployed machines is not possible. And it is precisely in these older machines where the benefits of successful self-monitoring solutions can have a bigger impact.

Therefore, there is an intrinsic interest in creating self-monitoring solutions using the faults happening in machines under normal usage. However, we are losing the main benefit of lab-condition testing: we do not necessarily have an indication of when the fault started happening in the machine. The closest we can get is to use the part sales and service logs, describing when a part or component was sold for a certain machine. This information would tell us when a failure happened in the machine, not when the fault started. In addition, these log systems have not been designed to define and analyze the fault that was corrected. There are necessary uncertainties:

- Uncertainties in the dates. Even assuming that the dates are logged correctly (which is not always the case due to human error or accounting policies), we know when a component was replaced in the machine, not when the failure happened.
- In fact, we do not know if the replaced component had failed. The replacement of a component can be due to a failure, but it can also be due to a preventive maintenance operation or a subjective decision by the technician.
- If there was a fault in the machine, we do not know if the component replacement solved it.
- If many components were replaced, we do not know which one was responsible for the fault or if different faults were happening simultaneously.
- We do not have the certainty that all the maintenance operations are recorded in the service logs. Some maintenance operations can be carried out by the staff operating the machines. In addition, users can buy services from different companies.
- In the case where the failure actually happened and the responsible component was replaced, we do not have information about when the fault started.

Using sale logs to label the faults is imprecise; we need to make the simplification of identifying component and type of fault, even though a given component can fail in different ways. This means that for the same type of fault, we may have different symptoms appearing in the monitoring data.

These uncertainties are common to many industrial endeavors. We are referring to traditional manufacturing companies that also provide after-sales service to the machines they produce. These companies possess knowledge about how the machines work and the data describing their operation and maintenance history. With recent technological developments, successful self-monitoring systems' benefits clearly outweigh their costs, making it interesting for traditional manufacturing companies to develop.

For broader industrial scenarios, there are still important characteristics that can be extrapolated from our studies: it is extremely interesting to monitor and gather data from machines operated by the end-users during real-life operations. However, obtaining labels for the faults in real-life operations can be challenging and costly.

Challenge II: learning with many possible faults and imperfect data

The next challenge, and the challenge that we devote most of the work carried out in this thesis, deals with the multi-fault nature of complex systems.

A sterilizer, a truck, an airplane, or a printer are complex machines composed of a great number of components. Each of those components can fail, i.e., each of those components can be associated with a different type of fault. Some simple components can be associated with just one type of fault; for example, a pipe is either leaking or not leaking. However, more complex components can have multiple types of faults associated with them, with their own symptoms and consequences.

In general, machines are well designed; they fail rarely. If we think about the lifetime of a machine, most fault types will rarely or even never happen. This means, on top of the challenges coming from the uncertainties arising from the repair logs, that we are going to have few failure cases for each type of fault, probably making our learned models' performance sub-optimal.

In addition, we are assuming that the quality of the monitoring data is not ideal, i.e., it will not be good enough to monitor every possible fault and component of the machine accurately. Machines such as trucks, cars, or sterilizers have been equipped with sensors for a while now, and that data has been recorded and saved. However, which signals to measure, what data to record, and how to process it has not been decided with the goal of developing self-monitoring systems. Usually, the reasons behind collecting this data are related to security or control.

The ideal situation described in Figure 1.2, degenerates into Figure 1.3. While in the ideal case, we could link every type of fault with a distinctive symptom in the recorded data, in a more realistic scenario, this will not happen. Some faults will not have any apparent symptoms in the data; others will have many different symptoms across different signals; groups of faults will produce the same or similar symptoms in all or in some signals.

The central assumption of our thesis is that in such complex machines, given the monitoring data and the label information at hand, it is going to be impossible to obtain accurate predictive models for each possible fault type. On the other hand, for a complex machine, just predicting that the machine as a whole will fail without further input for the technician carrying out the maintenance is not precisely useful. This is especially true for the case of predictions of future faults since the symptoms that are used to diagnose already broken machines can be very different from the symptoms that describe the degradation produced by an ongoing fault.

We assume that while it may not be possible to detect every possible fault with the monitoring data at hand, it will be possible to predict that a fault will happen within a reduced group of faults. For example, it might be impossible to learn a model to predict the failure of a valve in the vacuum system of

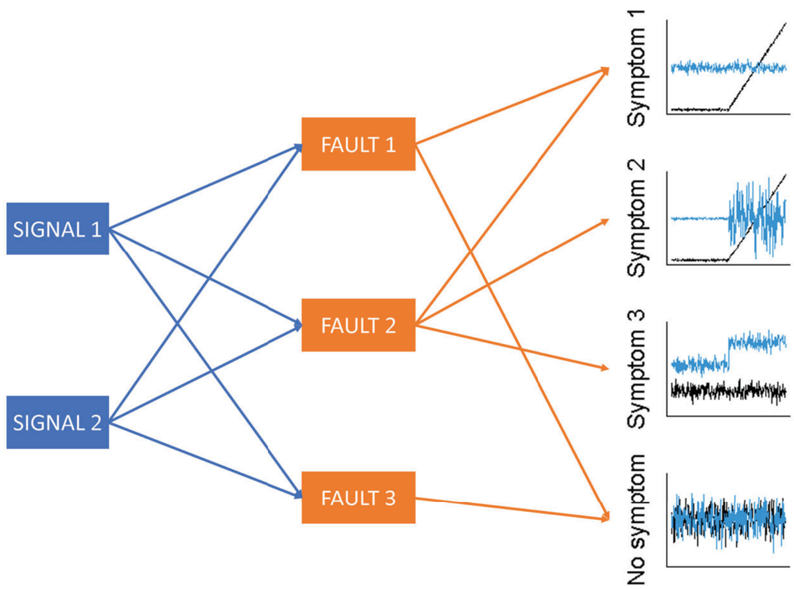


Figure 1.3: More realistic scenario for a self-monitoring solution.

a sterilizer. However, we might have enough information to learn a model successful at predicting a failure in the vacuum system. This assumption is, in turn, based on the fact that different faults can produce the same or similar symptoms in the monitoring data. In such a complex machine, there are going to be many different levels of similarity between faults.

We decide to encode these heterogeneous relations of similarity in hierarchies of faults. A hierarchy is a natural way to structure different levels of similarity: similar faults will be clustered together at the bottom of the hierarchy, while more dissimilar faults will be grouped close to the top of the hierarchy.

Our work in this thesis is devoted to extracting these similarity relationships in complex machines and exploiting them to improve the performance of systems. We will use hierarchical structures to find the balance between the ability to accurately predict the occurrence of a fault in the machine and the precision in determining which type of fault is responsible for the predicted failure.

1.3 Research questions

The underlying hypothesis of this thesis is that machine learning solutions employing hierarchies of faults can provide significant and measurable improvements over non-hierarchical approaches in the field of predictive maintenance. Based on this hypothesis, we formulate two sets of research questions, one set about the application domain and one set about the theoretical challenges:

- **AD-RQI** How do we obtain useful information from repair logs to train successful models?
- **AD-RQII** Can we use hierarchical approaches to improve fault diagnosis and prognosis tasks?
- **T-RQI** How do we develop methods to extract hierarchies from data?
- **T-RQII** What makes some hierarchies better than others?

1.4 Contributions

The main novelty of our work is to use a hierarchy of faults for predictive maintenance. Based on this work and the technical challenges it poses to the state-of-the-art machine learning techniques, we have found some important research gaps that we have explored.

To solve the problem of fault diagnosis, we have used extracted hierarchies of faults to train hierarchical multi-class classifiers. There does not exist a precise definition of what makes a good hierarchy. Practically, the best hierarchy is the hierarchy that maximizes a given classification performance. Intuitively though, a good hierarchy is the one that separates dissimilar objects at the top of the hierarchy, while the separation between very similar classes is done at the bottom.

However, it is not clear how to define the similarity between classes or how the quality of the hierarchy can affect the performance of a classifier trained on it. There are many different approaches to defining and measuring dissimilarities between classes, focusing on different aspects of the data and there are different ways to group the classes into a hierarchy. Measuring the quality of the hierarchy based on the performance of a classifier trained on it will undoubtedly be conditioned by how the classification is performed, which learning algorithm we use or how balanced the different binary classification problems are.

In this thesis, we explore in depth these concepts. In particular, we show how the intrinsic quality of a hierarchy becomes irrelevant to the performance of a classifier trained on it if the learning algorithm used is powerful. Which can lead to wrong evaluations of methods to extract hierarchies.

It is also not obvious what characteristics of the data make some hierarchies better than other. In a classification problem with four classes: "tiger", "lion", "frog", and "lizard"; it is intuitive to first distinguish between the groups "tiger" and "lion", and "frog" and "lizard". Then we can distinguish between the subgroups. It seems natural to think about this hierarchy as the best possible one, and that a hierarchical classifier trained on it, would be better than a classifier trained in any other hierarchy.

It seems natural that a hierarchical classifier trained on this hierarchy of classes should provide better results than on any other hierarchy. Our results and many others in the literature point in this direction. However, there is not an intuitive way to explain why an instance labeled with "frog" is correctly classified using this hierarchy, but with another hierarchy, it could be misclassified as "lion", "tiger", or "lizard".

In this thesis, we propose the concept of "heterogeneity of decision boundaries" to explain this phenomenon and why some hierarchies are, in fact, better than others.

To solve the problem of fault prognostics, we have used extracted hierarchies of faults to train multi-output classifiers. Our practical setup is special in that we train our models using monitoring data of a machine undergoing individual faults, but we are interested in predicting the time-to-failure for any of the components of a machine. The similarities between the fault patterns associated with different faults create new challenges that have not been treated before in the scientific literature.

The main contributions of this thesis are listed below:

	AD-RQ1	AD-Q2	T-RQ1	T-RQ2
Paper I	✔			
Paper II		✔		✓
Paper III		✔		✓
Paper IV			✔	
Paper V			✓	✔

Table 1.1: Contributions of the papers to different research questions,where ✔ signals the main contribution of the paper, ✓ signals a secondary contribution of the paper.

- We have created a methodology to filter events in the repair logs that is useful for predicting breakdowns (Paper I).

- We have applied hierarchical methods for fault diagnosis and fault prognosis that outperform their non-hierarchical counterparts (Papers II and III).
- We have identify common pitfalls in the literature and established best practices for evaluating methods to extract hierarchies for multi-class classification (Papers IV)
- We have identified the heterogeneity of decision boundaries as a key factor that can explain why classifiers trained some hierarchies perform better than others (Paper V).

1.5 Thesis Organization

The remainder of the thesis is organized as follows. Chapter 2 presents a summary of our contributions and results in the theoretical aspects of using hierarchical methods in supervised learning. Chapter 3 shows the application of these methods for fault diagnosis and fault prognosis. Chapter 4 gives summary of the papers included in this thesis. Finally, Chapter 5 concludes the thesis by providing an overall summary of our contributions.

Chapter 2

Hierarchies

In Chapter 1, we have introduced the concept of hierarchies of faults for self-monitoring systems. Our central hypothesis is that machine learning models trained on extracted hierarchies will bring measurable benefits.

In this chapter, we focus on the use of hierarchies for supervised methods in machine learning and my contributions to the state of the art. In particular, we address research questions **T-RQI** and **T-RQII**, i.e., "how do we develop methods to extract hierarchies from data?" and "what makes some hierarchies better than others?".

Hierarchies of classes for supervised methods have been extensively used in the literature, in particular in the fields of multi-class [19] and multi-label classification [14]. In a lesser extent, they have also been employed in multi-output regression [11]. In the latter case, instead of having classes at the leafs of the tree, we have targets for which we need to predict a numerical value.

In the literature, there is an ambiguous definition of "hierarchical classification." It can be referred to problems where a hierarchy of classes is already established [18]. Frequently, in addition to the classes at the leaves of the hierarchy, the meta-classes at the intermediate nodes of the hierarchy have its own meaning, and obtaining good performance at predicting those meta-classes is important. Hierarchical classification has also been used for "flat" multi-class problems where no hierarchy is given, and a hierarchy is extracted and employed [1]. In these later case, the ultimate goal is to still obtain good classification performance in the original classes, placed at the leaves of the tree.

We can define a hierarchy as a set of nodes that are iteratively merged, forming a tree structure with intermediate nodes. Often, classification algorithms are designed an optimized to deal with binary classification problem. This is why, typically, hierarchies used for classification are restricted to have only two children per parent node. However, that is not a real restriction and there have been interesting contributions using other types of hierarchies [16].

In this thesis, we are interested in exploring how to extract hierarchies from the data when no hierarchy is given, i.e., finding groups of classes that allow building better models. We are also focusing on hierarchies where only binary splittings are allowed.

2.1 Hierarchies: extraction

Formally, a hierarchy is defined as a binary dendrogram. Starting from the root, all classes are iteratively separated into two sub-trees until each class is placed in its own leaf of the tree. An example of such a hierarchy can be seen in Figure 2.1.

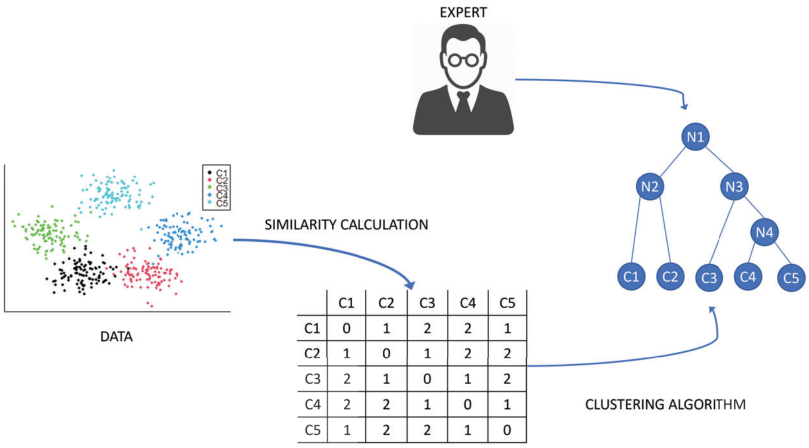


Figure 2.1: Diagram representing how to extract hierarchies. A hierarchy can be defined by an expert, or extracted from the data in two steps: measuring the dissimilarity and clustering the classes

In the absence of a predefined hierarchy, we are interested in extracting hierarchies from the data. This is achieved in two steps 2.1: first we need to measure the dissimilarities between classes, each of them characterized by the points labelled with the corresponding class; then we need to use a clustering algorithm to create the hierarchy.

2.1.1 Dissimilarity

A class is defined by a set of points annotated with the same label. Evaluating the dissimilarity between two classes c_1, c_2 means evaluating a function of the form $d(c_1, c_2) = f(D_{c_1}, D_{c_2})$, where $D_c = \{\mathbf{x_c^i}\}$ is the distribution of points defined by the feature vectors $\mathbf{x^i}$ annotated with label c_i.

Algorithm 1 Agglomerative clustering

1. Begin with n clusters, each of them corresponding to one fault
2. Compute distance between every pair of clusters.
3. Find the most similar pair of clusters and merge them.
4. Calculate the distance of the newly created cluster to the rest.
5. Repeat steps 3 and 4 until there is just one cluster.

Representative Based Distance One way to measure the dissimilarity between classes consists of obtaining a representative for each class and evaluating their distance. This approach is frequently employed using the centroid as the class representative [7]. Once a single data point per class is chosen, any distance metric can be used to calculate the dissimilarity. It is a straightforward approach and usually quite favorable in terms of computational complexity. However, it presents certain drawbacks. For example, the point chosen for a given class may not be representative or the features used to measure the distance that discriminate between fault might be hidden in the presence of multi-dimensional noise.

Classifier Based Dissimilarity The intuition behind using classifier based distance [5] is that if a classifier can not distinguish clearly between two classes, those classes must be similar. Conversely, if a classifier can clearly differentiate between two classes, those two classes must be dissimilar. The final measure of dissimilarity between two classes is then the performance of a classifier trained to distinguish between them.

Algorithm 2 Divisive clustering

1. Begin with all faults in one cluster
2. Divide the objects of each cluster into two groups
3. Repeat until all faults are placed in a cluster of its own.

2.1.2 Hierarchical clustering

Once we have evaluated the similarities between classes, we need to cluster them together, to finally obtain the hierarchy. There are many clustering methods present in the literature [17]. Without getting into details, we group these methods in two classes: agglomerative clustering 1 and divisive clustering 2.

	Dataset name	Instances	Attributes	Classes	Min	Max
1	Collins	1000	19	30	6	80
2	Pen digits	10992	16	10	1055	1144
3	Abalone	4168	9	21	6	689
4	Arrythmia	438	262	9	9	245
5	Image Segmentation	2310	18	7	330	330
6	Cartdiotocography	2126	35	10	53	579
7	Semeion	1593	256	10	155	162
8	Texture	5500	40	11	500	500
9	Statlog(Sat)	6430	36	6	625	1531
10	Walking	149332	4	22	911	21991
11	Usps	9298	256	10	708	1553
12	Bach	5571	90	65	6	503
13	Letters	20000	16	26	734	813
14	Helena	65196	27	100	111	4005
15	Plants(margin)	1600	64	100	16	16
16	Plants(shape)	1600	64	100	16	16
17	Plants(texture)	1599	64	100	15	16
18	Mnist	70000	719	10	6313	7877
19	Covertype	581012	54	7	2747	283301

Table 2.1: Information about datasets used in this section: number of instances, number of features, number of classes, and number of instances for the most and lest frequent class.

2.2 Evaluation of extracting methods of hierarchies for multi-class classification

Extracting of hierarchies for multi-class classification problems has been extensively used in the literature for many different applications. However, a rigorous evaluation of the methods to extract hierarchies is usually missing. In this section, we present a set of experiments showing some common pitfalls in the evaluation of methods to extract hierarchies for multi-class classification. We use the datasets shown in Table 2.1, to showcase our conclusions.

2.2.1 Comparing hierarchical methods with non hierarchical methods.

Many contributions to the literature on extracting hierarchies to decompose the multi-class classification problem consist of proposing a method to extract the hierarchy, evaluating the classifier trained on the hierarchy, and compare with non-hierarchical methods. Comparing non-hierarchical methods for multi-class classification with hierarchical methods can lead to incorrect conclusions,

	Single Classifier	Random	Informed
5	$\underline{0.926 \pm 0.0054}^{\downarrow\downarrow}$	0.9537 ± 0.0031	0.9617 ± 0.0046
9	$\underline{0.8383 \pm 0.0061}^{\downarrow}$	0.8627 ± 0.0033	0.8644 ± 0.0047
10	$\underline{0.5099 \pm 0.0013}^{\downarrow\downarrow\downarrow}$	0.599 ± 0.0017	0.6028 ± 0.0011
11	$\underline{0.8059 \pm 0.0031}^{\downarrow\downarrow\downarrow}$	0.8551 ± 0.006	$\mathbf{0.8929 \pm 0.0032}^{\uparrow\uparrow\uparrow}$
14	$\underline{0.20832 \pm 0.00091}^{\downarrow\downarrow\downarrow}$	0.2361 ± 0.0038	$\mathbf{0.2685 \pm 0.0023}^{\uparrow\uparrow\uparrow}$
19	$\underline{0.8184 \pm 0.0031}^{\downarrow\downarrow\downarrow}$	0.8595 ± 0.0045	$\mathbf{0.8784 \pm 0.0014}^{\uparrow\uparrow}$

Table 2.2: Comparison of single classifier, Hierarchical Multi-class Classification using a random hierarchy, and using an informed hierarchy. Underlined are those results significantly worse than the other two, highlighted those that are significantly better than the rest. One arrow indicates a p-value smaller than 0.05; two arrows, 0.01; three arrows, 0.001.

leading to the validation of the method to extract the hierarchy, where in fact the extracted hierarchy is not useful.

In Table 2.2, we can see the results of using a single multi-class classifier, a hierarchical classifier trained on a randomly generated hierarchy and a hierarchical classifier trained on a hierarchy extracted using the euclidean distance between class-centroids as dissimilarity measure and an agglomerative clustering algorithm. As learning algorithm, we use the CART algorithm, where we have tuned the pruning parameter and the size of the leaves.

In datasets 5,9, and 10, we observe significant differences between the single classifier and both hierarchical approaches. But, we do not observe significant differences between the hierarchical classifiers using a randomly generated hierarchy and the informed hierarchy.

Simply comparing a hierarchical approach with a non-hierarchical approach can falsely lead us to conclude that the method to extract hierarchies is useful We can avoid this by using random hierarchies as a benchmark. Suppose the extracted hierarchy is not significantly better than the random hierarchy but is significantly better than the non-hierarchical approach, like in datasets 5,9, and 10. In that case, we can conclude that there exist inherent advantages to using any hierarchy, but we can not conclude that the extracted hierarchy is good. However, in cases like datasets 11, 14, and 19, when the hierarchical classifier using an informed hierarchy is significantly better than the classifier trained on the random hierarchy, we can conclude that some inherent improvement comes from the hierarchy extraction method.

2.2.2 Evaluating methods to extract hierarchies using different base learners

We are trying to evaluate methods to extract hierarchies based on the classification performance of classifiers trained on them. In order to be rigorous about it, we need to study the effect of using different base learners in the nodes of the hierarchies.

To showcase the effects of the power of the base learner, we are going to use three different learning algorithms: CART (tuning the pruning parameter and the number of instances in the tree's leaves), logistic regression, and SVM with Gaussian kernel (tuning the width of the kernel and the regularization parameter). We train hierarchical classifiers on 100 randomly generated hierarchies and evaluate the accuracy and its variance for each hierarchy using a 20-fold Monte-Carlo validation method.

	Correlations			$\overline{\sigma_h}/\sigma(h)$		
	SVM-GLM	SVM-CART	GLM-CART	SVM	CART	GLM
5	-0.08	-0.18	0.06	1.02	0.85	0.20
7	-0.03	0.02	$0.52^{\uparrow\uparrow\uparrow}$	0.86	0.61	0.42

Table 2.3: On the left, pairwise correlations between different learning algorithms trained on the same hierarchies. Three arrows indicate that the correlation is significantly bigger than 0, with a p-value smaller than 0.001. On the right, ratio between the mean variance for each individual hierarchy and the variance of the performances across all hierarchies.

On the right side of Table 2.3, we can read the ratio between the average of the variances for each of the 100 sampled hierarchies over the variance of all the accuracies. This measure indicates how different hierarchies affect the hierarchical classifiers trained with the different learners. Values close to 1 mean that the choice of the hierarchy has no significant effect on the performance of the classifiers trained on them. Values close to 0 indicate that the choice of the hierarchy has measurable effects.

In the case of these two datasets and three learning algorithms, we can see that the SVM has, in general, high values close to 1, indicating this learning algorithm is powerful enough not to be affected by the choice of the hierarchy. On the other side of the spectrum, we see the logistic regression, where the values are closer to 0, indicating the importance of selecting the best possible hierarchy to obtain the best possible classification performance.

For a powerful enough classifier, the quality of the hierarchies used has no measurable effect on the final performance of the hierarchical classifier; at each node, the classifier is taking optimal decisions independently of how the classes are split. Therefore, the classifier performance cannot be universally used to measure the quality of the hierarchy since the quality of the learner can mask it.

2.2.3 Good hierarchies are never bad hierarchies

It is worth asking the question "is there an inherent quality of the hierarchy?". Posed like this, the question might be to broad. However, we can answer the question "is the same hierarchy good for different base learners?".

On the left side of Table 2.3, we can read the pairwise correlation of the performance of the three different learning algorithms presented in the previous subsection trained on the same hierarchies. We only observe a significant correlation on dataset 7, where both the CART and the logistic regression algorithms showed sensitivity to the quality of the hierarchy. A positive correlation and bigger than 0 means that there exist an inherent quality to the different hierarchies, that helps both learning algorithm.

In the rest of the cases, we do not observe significant correlations between the performance of different learning algorithms. But this does not necessarily means that there is no inherent quality of the hierarchies, it simply means that at least one of the classifiers is powerful enough to not be affected by it.

In Table 2.4, we can observe further proof of the relative quality of the hierarchies. We train hierarchical classifiers with three base learners (CART, XGBoost, and SVM) on the same set of hierarchies (randomly generated and extracted after measuring the euclidean distance between the centroid of each class and using agglomerative clustering algorithm).

There are datasets where there is no significant difference for any base learners. We can conclude that these datasets are too simple for all of the base learners, rendering the quality of the extracted hierarchies irrelevant, e.g., datasets 1, 3, 5, 7, 8, and 9.

There are significant differences between the different learning algorithms for the rest of the datasets. In general, SVM outperforms XGboost, which, in turn, outperforms CART. One important result to focus on is that whenever the informed hierarchy results in significantly improved accuracy compared to the random hierarchy for one learning algorithm for a given dataset, and there is no significant difference for another dataset, the significant difference always happens for the weaker learner. This result further points in the direction that we are, in fact, extracting useful hierarchies, but for a powerful enough base learner, the quality of the extracted hierarchy is irrelevant.

2.2.4 Classifier based distance is a spectrum

So far, we have only extracted hierarchies using the euclidean distance between the centroids as the dissimilarity between classes. Classifier-based distance uses the intuition that if two classes are similar, a classifier trained to separate them will have low performance, while if the classes are dissimilar, the performance will be high.

In Figure 2.2, we can observe the evolution of the performance of a hierarchical classifier using logistic regression as a base learner trained on different

	CART		XGBoost		SVM	
	Random	RBD	Random	RBD	Random	RBD
1	0.1409±0.0096	0.171±0.011	0.257±0.016	0.267±0.018	0.304±0.061	0.305±0.049
2	0.9485±0.0023	**0.9551±0.002**$^{\uparrow}$	0.9895±0.0014	0.9908±0.0011	0.9940 ± 0.0020	0.9940 ± 0.0020
3	0.2417±0.0076	0.2246±0.0084	0.262±0.011	0.253±0.011	0.270 ± 0.018	0.274±0.021
4	0.731±0.016	**0.766±0.017**$^{\uparrow}$	0.762±0.018	0.767±0.017	0.726±0.053	0.737±0.048
5	0.9537±0.0031	0.9617±0.0046	0.9784±0.0031	0.9823±0.0038	0.974±0.0074	0.978±0.0087
6	0.9990±0.0006	0.9995±0.0005	0.9995±0.00058	0.9998±0.0004	1±0	1±0
7	0.731±0.016	**0.787±0.011**$^{\uparrow\uparrow}$	0.9224±0.0067	0.9328±0.0071	0.953±0.12	0.953±0.015
8	0.9242±0.0037	0.9333±0.0026	0.9814±0.0017	0.9835±0.0017	0.998±0.002	0.999±0.0015
9	0.8627±0.0033	0.8644±0.0047	0.9186±0.0029	0.9192±0.0032	0.9220 ±0.0066	0.9170±0.0094
11	0.8551±0.006	**0.8929±0.0032**$^{\uparrow\uparrow\uparrow}$	0.9603±0.0024	**0.9653±0.003**$^{\uparrow\uparrow}$	0.970±0.0076	0.976±0.0053
12	0.7242±0.0044	**0.7473±0.0067**$^{\uparrow\uparrow}$	0.7844±0.0064	0.7907±0.0062	0.811±0.014	0.816±0.014
13	0.8541±0.0021	**0.8633±0.0028**$^{\uparrow}$	0.9562±0.0016	0.9568±0.0018	0.962±0.0048	0.968±0.0029
14	0.2361±0.0038	**0.2685±0.0023**$^{\uparrow\uparrow\uparrow}$	0.3582±0.0017	**0.3673±0.0017**$^{\uparrow\uparrow\uparrow}$	0.34±0.0051	**0.356±0.0044**$^{\uparrow\uparrow\uparrow}$
15	0.348±0.012	**0.498±0.019**$^{\uparrow\uparrow\uparrow}$	0.64±0.025	**0.716±0.012**$^{\uparrow\uparrow\uparrow}$	0.814±0.035	0.836±0.037
16	0.323±0.01	**0.4015±0.0097**$^{\uparrow\uparrow\uparrow}$	0.509±0.019	0.544±0.016	0.713±0.040	0.716±0.038
17	0.401±0.015	**0.513±0.012**$^{\uparrow\uparrow\uparrow}$	0.668±0.018	**0.715±0.015**$^{\uparrow\uparrow\uparrow}$	0.844±0.030	0.850±0.028

Table 2.4: Comparison of Hierarchical Multi-class Classification trained on a random hierarchy and on a hierarchy using Representative Based Dissimilarity using CART, Extreme Gradient Boosting, and SVM using gaussian kernel. Highlighted are the results that are significantly better than the rest. One arrow indicates a p-value smaller than 0.05; two arrows, 0.01; three arrows, 0.001.

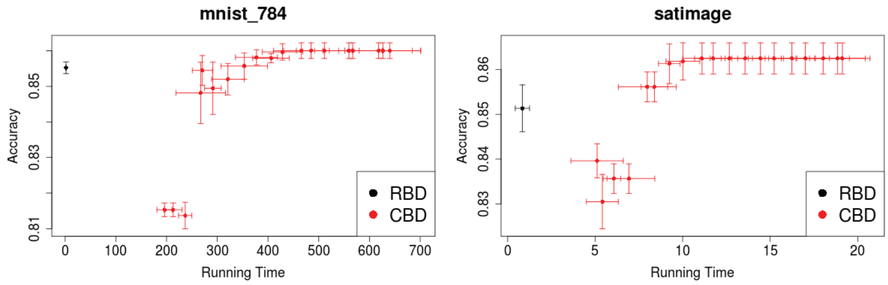


Figure 2.2: Evolution of the performance of Hierarchical Multi-class Classification trained on different hierarchies as the complexity of the CBD classifier increases, compared against RBD.

hierarchies. Each hierarchy has been computed using classifier-based distance using CART as a classifier. By tuning the pruning parameter of the CART classifier, we control the complexity of the model. As the classifier used to measure the dissimilarity becomes more complex (using the time it takes to train the classifier as a proxy for the complexity), the performance of the hierarchical classifier trained on the extracted hierarchy improves. As a benchmark, we have kept the performance of the classifier trained on the hierarchy extracted using the euclidean distance between centroids.

The intention behind using CBD is to measure how easy two classes are to separate. More specifically, we want to know how easy it will be for the learning algorithm trained on the hierarchy to separate between these two sets of classes. However, the results of training the CBD classifier give us only an approximation of how well this base classifier will perform.

With this motivation in mind, in the following experiment, we want to explore the relationship between the power of the CBD classifier used to measure dissimilarities between classes and the power of the base classifier trained on the hierarchy.

Figure 2.3 shows how the classification accuracy across two different datasets changes depending on choosing different configurations of the complexity parameters for both the CBD classifier and the base classifier. The most interesting results can be seen for the **Letters** dataset: for the simpler base classifier, the accuracy drops as the classifier-based distance classifier gets more complex. The same behavior cannot be seen for the **MNIST**. This shows that the relationships between classes found by a powerful classifier based distance classifier will not necessarily be useful when we have a weak base classifier.

In both examples in Figure 2.3, the relevance of the quality of the hierarchy decreases as the quality of the base classifier increases. This confirms our conclusion that a powerful enough base classifier is insensitive to the quality of the hierarchy.

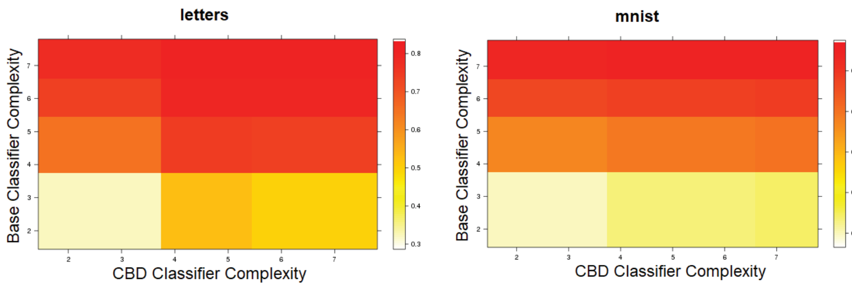


Figure 2.3: Comparison of the complexity of the CBD classifier and the base classifier

Classifier-based distance has previously been used in several places in the literature (e.g., [19], [5]). It is assumed that a simple classifier can be used to measure classifier based distance. However, we have shown that this is not necessarily the case. Classifier-based distance depends significantly on the classifier used to measure it, and as such needs to be understood and used.

2.3 Underlying reasons to explain why some hierarchies work better than others

We have observed in the previous section that different hierarchies can affect the performance of a hierarchical classifier trained on them. There is however an unanswered question: why? Some instances are correctly classified with one decomposition of the problem, while if the decomposition is different, they are not correctly classified.

2.3.1 Classification: Heterogeneity of decision boundaries

One possible reason that can explain this behavior is the heterogeneity of decision boundaries. We define a decision boundary, as the hyper-surface that best separates two sets of classes. With this definition, solving a multi-class classification problem is equivalent to finding all the one-vs-one decision boundaries.

The reason why some decomposition methods work better than others then, rely on how these one-vs-one decision boundaries are grouped. In the traditional one-vs-all approach, one classifier is trained for every class to separate them from the rest of the classes, effectively solving all the one-vs-one decision boundaries associated with said class. In the also traditional one-vs-one case, all individual decision boundaries are learn independently.

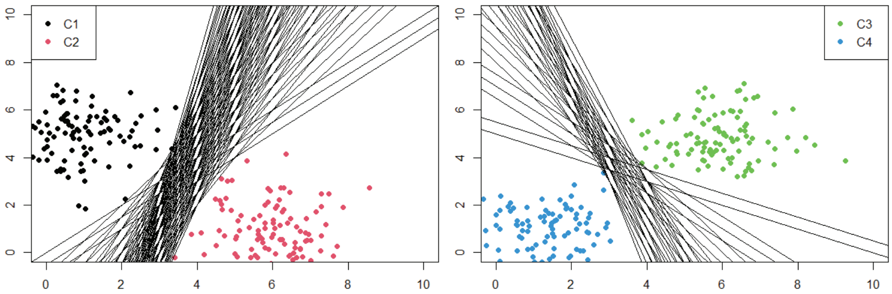


Figure 2.4: Toy examples to illustrate the concept of heterogeneity of decision boundaries. In both cases, we have two sets of points labelled with different classes and the set of possible lines that could perfectly separate them.

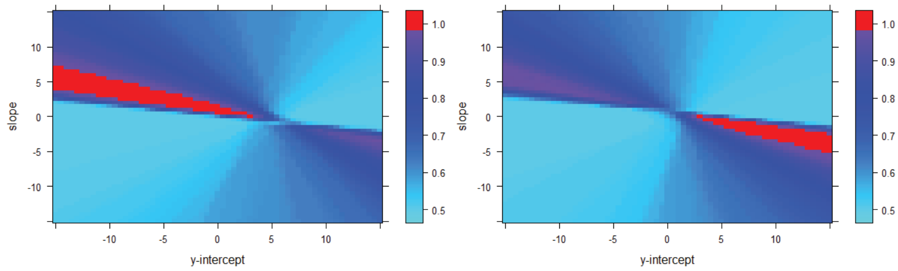


Figure 2.5: Heatmap of the accuracy of the line classifiers trained to separate the corresponding classification problems in Figure 2.4

Parametric algorithms: toy example To illustrate the concept of heterogeneity of decision boundaries, we propose a simple toy example in Figure 2.4. In it, we have two pairs of classes and all the straight lines that could be used as decision boundaries and have a perfect accuracy at separating them. If we wanted to find a straight line, that would simultaneously separate C1 from C2, and C3 from C4, this line could not achieve a perfect separation.

This phenomenon can be further visualized in the parameter space. In Figure 2.5, we can see the accuracy heatmaps of the line classifiers solving the corresponding classification problems in Figure 2.4. Each line can be defined by two parameters: its y-intercept and its slope. There is no intersection between the regions of maximum performance of these heatmaps, meaning that there is no single straight line that could be used to simultaneously classify accurately classes C1 and C3, against classes C2 and C4.

Non-parametric algorithms: hyper-parameter tuning The previous experiment was realized by trying different lines as decision mechanism. An algorithm that would find the best possible line is logistic regression. Logistic regression is a parametric algorithm, i.e., it fits the y-intercept and the slope value to minimize a given error function.

The description of the decision boundaries as the characterization of the learning algorithm's performance in the parameter space is key to our definition of heterogeneity of decision boundaries. But what happens with non-parametric learning algorithms?

Algorithms such as SVM, decision trees, and KNN are defined as non-parametric learning algorithms. So, in principle, our definition of heterogeneity of decision boundaries is not applicable to them.

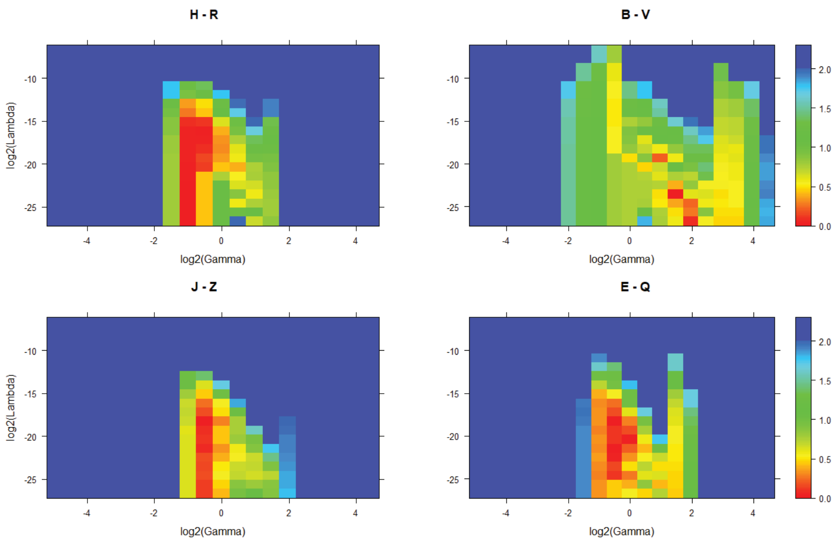


Figure 2.6: Visualization of the relative classification performance of different hyper-parameter configurations for four different decision boundaries. The color gradient corresponds to the relative difference in terms of standard deviations from the best possible performance. On the top row, we have an example of heterogeneous decision boundaries, where the regions of maximum performance do not overlap. On the bottomo row, an example of two decision boundaries that are homogeneous, i.e., their regions of maximum performance overlap.

However, these learning algorithms are not applied "non-parametrically" in real-life tasks. Instead, there is usually a phase of hyper-parameter tuning. In the case of KNN, we need to select the number of nearest neighbors; in the case of SVM, we need to choose an appropriate kernel and regularization

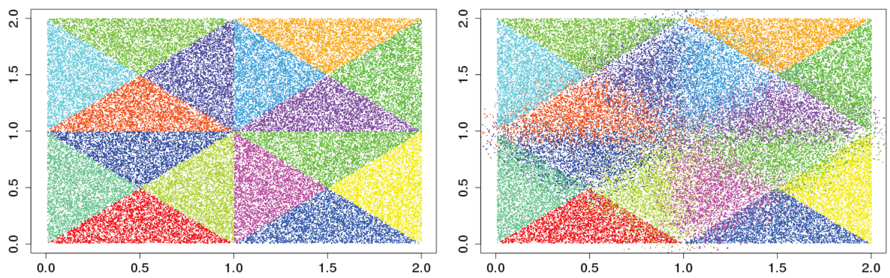


Figure 2.7: On the left, the training set of the first iteration (no noise). On the right, the training for the last iteration with noise in all inner triangles.

parameter; in the case of decision trees, we can choose the pruning parameter of the number of instances in each split.

The concept of the heterogeneity of decision boundaries, in these cases, is defined in the hyperparameter space. There can be a set of parameters that prove best to solve a given decision boundary, but they may not be the best for another decision boundary. We define two decision boundaries to be heterogeneous if their regions of maximum performance in the hyper-parameter space do not overlap.

In Figure 2.6, we can see an example of two set of heterogeneous and homogeneous decision boundaries from a real dataset (**Letters**) using SVM with gaussian kernel, tuning the width of the kernel and the regularization parameter.

Results Equipped with the concept of heterogeneity of decision boundaries, we decide to study the different methods to decompose multi-class classification problems. For this experiment, we will use CART as learning algorithm and we will tune the pruning parameter.

The key tool to manipulate the heterogeneity of decision boundaries of our dataset is based on understanding how the decision tree can solve diagonal and parallel to the axis decision boundaries in the presence of noise. To accurately solve a diagonal decision boundary, we need a very big tree since the CART algorithm only makes decisions parallel to the axes. But, very deep trees are penalized when solving parallel to the axis decision boundaries in the presence of noise, since they will overfit. We use this intuition to create a synthetic multi-class dataset where we can tune the heterogeneity by adding noise to some of the decision boundaries. In Figure 2.7, we have an example of the dataset used without noise (and homogeneity) and with noise in all the inner triangles (full heterogeneity).

In Figure 2.8, we can observe the performance of the three basic approaches to solving multi-class classification problems: a single classifier, one-

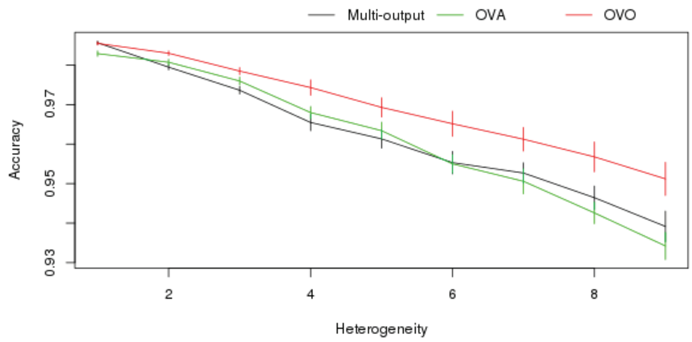


Figure 2.8: Accuracy comparison of multi-output, OVA, and OVO classifiers as the heterogeneity grows.

vs-all (OVA), and one-vs-one (OVO). Without noise (without heterogeneity), there are no significant differences between the approaches. When decision boundaries are fully homogeneous, the performance of the multi-output classifier is indistinguishable from OVO, and only slightly higher than OVA. As we inject more and more heterogeneity through noise, the performance of all the classifiers degrades, obviously. However, both OVA and the multi-output classifier suffer more and perform increasingly worse than OVO.

The key observation here is that OVO treats every decision boundary individually, finding the optimal hyperparameter configuration for each. Thus, the only difference in performance comes from the problem becoming inherently harder with more noise. The other two approaches group together decision boundaries, including heterogeneous ones, which necessarily leads to additional performance loss.

We now introduce two ad-hoc approaches to solve the multi-class problem by decomposing it using the knowledge of heterogeneity of decision boundaries (see Figures 2.9 and 2.10). The first one, minimal decomposition, uses multi-output classifiers at the nodes, the second one Hierarchical Aware Hierarchy, uses binary splits in the hierarchy.

In Fig. 2.11, we can see the comparison of the two approaches, together with OVA, with respect to OVO. Again, those decompositions that separate decision boundaries into homogeneous groups obtain classification performances similar to OVO, and significantly outperform approaches that do not consider this information, like OVA.

Measuring heterogeneity of decision boundaries and exploiting it. As an example, we are going to use again the letter dataset. The dataset has 20 000 instances and 16 features consisting of descriptors of distorted letters from different fonts, corresponding to the 26 letters in the English language.

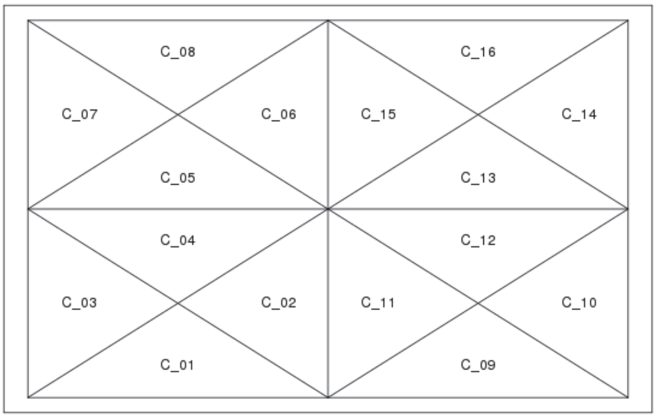


Figure 2.9: Labels for each of the classes in Figure 2.8

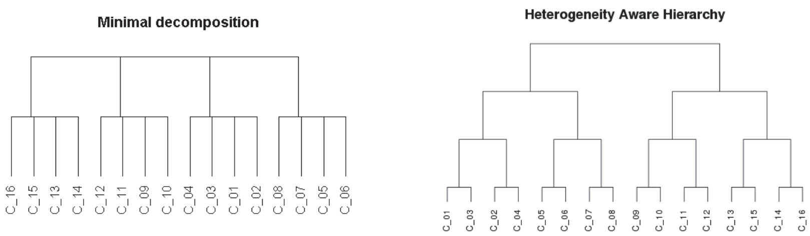


Figure 2.10: Decomposition of the original multi-class problem into a hierarchy of classes.

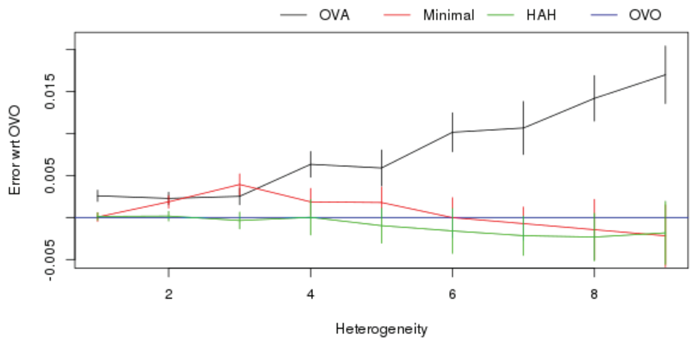


Figure 2.11: Error comparison of OVA, minimal decomposition, and hierarchical decomposition with respect to OVO.

For each decision boundary, we are going to compare the accuracy of a classifier with different hyper-parameter configurations. Let A^i_S be the array of accuracies of a family of classifiers solving the decision boundary i over the hyper-parameter space S. The plots in Figure 2.6 are, in fact, the visualization of different A^i_S, where we have sampled a grid of hyper-parameter configurations. We measure the amount, or degree, of heterogeneity between two decision boundaries i and j with the equation:

$$HDB(i,j) = max(A^i_S) + max(A^j_S) - max(A^i + A^j)_S \tag{2.1}$$

The last term of the equation corresponds to the selection of the hyper-parameter configuration that produces the highest accuracy of both decision boundaries as they are combined and solved together. If the two decision boundaries share a region of the hyper-parameter space where they achieve the maximum accuracy, the value of HDB is 0, i.e., the decision boundaries are homogeneous. If they do not share the region of maximum performance, the maximum value of the combined $(A^i + A^j)_S$ will not coincide with the individual maxima of A^i_S and A^j_S. In this case, the value of HDB would be bigger than 0, indicating that the decision boundaries are heterogeneous and capturing the degree of heterogeneity. This measure can be easily expanded to more than two decision boundaries:

$$HDB(1,..,n) = \sum_{i=1}^{n} \max(A^i_S) - \max(\sum_{i=1}^{n} A^i)_S \tag{2.2}$$

Using the measure of the heterogeneity of decision boundaries, we are going to extract a hierarchy in a top-down manner by selecting at each step the split with the lowest value for HDB. To have a benchmark, we compare the results against a randomly sampled hierarchy and OVO. Results are presented in Table

One-vs-One	Random hierarchy	HDB hierarchy
0.9753 ± 0.0054	0.9692 ± 0.0039	0.9737 ± 0.0040

Table 2.5: Accuracies of One-vs-One, hierarchical classification using a random hierarchy, and hierarchical classification using HDB measure to create the hierarchy.

2.5. We perform paired t-test comparisons between the different approaches. The hierarchical approach using HDB measurements to create the hierarchy is significantly better than the hierarchical approach using a random hierarchy, with a p-value of 0.04. There is no significant difference between the One-vs-One approach and the hierarchy using HDB (with a p-value of 0.22).

2.3.2 Hierarchies for Regression: the problem of similar patterns

Based on our application domain, we are trying to train multiple regressors to predict time-to-failures related to different types of fault for the same machine. In the standard formulation of multi-output regression, the task is to train a model $f : \mathcal{X}^p \longrightarrow \mathcal{Y}^d$, where $\mathcal{X}^p$ is the feature space of dimension p, and $\mathcal{Y}^d$ is the output space of dimension d.

In our case study, we have to learn models for time-to-failure prediction for different types of faults. This is a particular case, since each instance is labeled with the time-to-failure for a single fault.

There are two possible approaches to labeling the data to train, illustrated in Figure 2.12. If we denote by S, each of the sequences of time-to-failure, we can choose to train individual models for each output trained only using the corresponding sequence; we name this approach "individual model." On the other hand, we can also use the rest of the sequences corresponding to other faults and label them with the maximum time-to-failure; we name this approach "contrast model."

Individual models: the curse of cross-predictions. In Fig. 2.13, we can visualize the data corresponding to three different sets of sequences, the failure happens at the end of the sequence and the rest of the data is labelled accordingly. We could create a model for fault 6, using the *Stream9f* feature, obtaining a model that would tell us when the fault is going to happen with a reasonable accuracy (notice how the data flattens towards the end), and clearly distinguish it from the other faults. However, without using the data from other faults as contrast, it seems better to use the feature *Stream9b*; this feature has a monotonic growth towards the occurrence of the failure. This

		y_1	y_2	y_3
S1	x_1	4	-	-
	x_2	3	-	-
	x_3	2	-	-
	x_4	1	-	-
	x_5	0	-	-
S2	x_6	-	4	-
	x_7	-	3	-
	x_8	-	2	-
	x_9	-	1	-
	x_{10}	-	0	-
S3	x_{11}	-	-	4
	x_{12}	-	-	3
	x_{13}	-	-	2
	x_{14}	-	-	1
	x_{15}	-	-	0

		y_1	y_2	y_3
S1	x_1	4	T_max	T_max
	x_2	3	T_max	T_max
	x_3	2	T_max	T_max
	x_4	1	T_max	T_max
	x_5	0	T_max	T_max
S2	x_6	T_max	4	T_max
	x_7	T_max	3	T_max
	x_8	T_max	2	T_max
	x_9	T_max	1	T_max
	x_{10}	T_max	0	T_max
S3	x_{11}	T_max	T_max	4
	x_{12}	T_max	T_max	3
	x_{13}	T_max	T_max	2
	x_{14}	T_max	T_max	1
	x_{15}	T_max	T_max	0

Figure 2.12: Two different strategies to label each time-to-failure sequence of data. On the left, we choose to only learn the models for the corresponding fault. On the right, we use the data from the other faults as contrast.

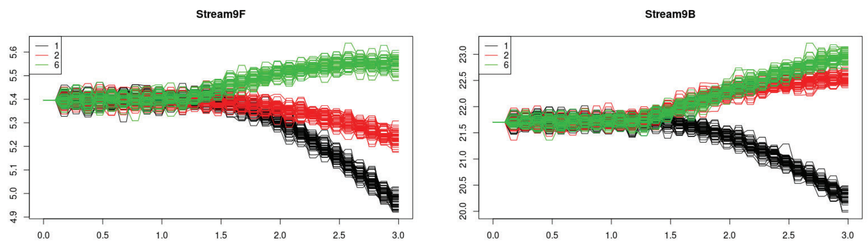


Figure 2.13: Example of two features corresponding to three different types of faults. In Stream9F, there is a clear distinction between the 3 types of faults. In Stream9b, however, faults 2 and 6 are harder to distinguish.

model will create many false alarms predicting time-to-failures for fault 6 in cases where it was actually fault 2 happening.

	CPF	FAH	FII
Independent	1 ±0	0 ±0	0.40±0.01

Table 2.6: Results of the individual model. CPF indicates the correctly predicted failures, FAH indicates the false alarms when there is no fault in the machines, and FII measures the false alarms when the fault is incorrectly identified.

In Table 2.6, we can observe an evaluation of the performance of the individual model. The evaluation consists on detecting a fault when the predicted time-to-failure goes under a selected threshold. The independent model is able to recognize every fault, but it also issues a lot of alarms predicting the wrong type of failure. Individual models suffer in the presence of similar patterns labeled with different faults, they result in incorrect predictions, predicting low time-to-failures for the incorrect type of fault.

Contrast models: the curse of similar patterns In the contrast model approach, we train an individual regressor for each type of fault, but we also use the data corresponding to other faults as contrast, in order to try to overcome the drawbacks of the individual models.

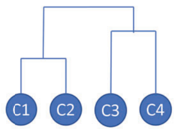
	CPF	FAH	FII
Contrast	0±0	0±0	0±0

Table 2.7: Results of the contrast model in the presence of similar patterns. CPF indicates the correctly predicted failures, FAH indicates the false alarms when there is no fault in the machines, and FII measures the false alarms when the fault is incorrectly identified.

However, this type of approach is very vulnerable to the presence of similar patterns annotated with different labels. In the case of full similarity, this means that we are training a regressor using completely different labels for fully similar instances, making the models fail to predict any failure. In Table 2.7, we can observe the results of evaluating the contrast model in the presence of fully similar faults. Not a single fault was predicted.

Flat approaches for multi-output regression suffer from clear drawbacks. Regressors only trained on their corresponding target risk recognizing patterns associated with other targets, resulting in our application domain in false alarms. Regressors trained using the rest of the targets as contrast risk being completely useless if similar patterns appear in different targets. These

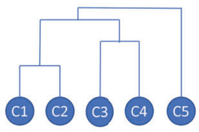
	C1	C2	C3	C4
C1	0	0.55	0.9	0.9
C2	0.55	0	0.9	0.9
C3	0.9	0.9	0	0.8
C4	0.9	0.9	0.8	0



MERGE	HEIGHT
C1 U C2	0.55
C3 U C4	0.8
{C1, C2} U {C3,C4}	0.9

Figure 2.14: Hierarchy extracted for four classes using the dissimilarity metric. We use agglomerative clustering with average criterion.

	C1	C2	C3	C4	C5
C1	0	0.55	0.9	0.9	1
C2	0.55	0	0.9	0.9	1
C3	0.9	0.9	0	0.8	1
C4	0.9	0.9	0.8	0	1
C5	1	1	1	1	0



MERGE	HEIGHT
C1 U C2	0.55
C3 U C4	0.8
{C1, C2} U {C3,C4}	0.9
{C1, C2, C3, C4} U C5	1

Figure 2.15: Hierarchy extracted for five classes using the dissimilarity metric. We use agglomerative clustering with average criterion.

two phenomena are very obvious and apparent with our problem setup since we have disjoint labeling of the different targets during the training phase. However, the underlying characteristics related to the similarity and dissimilarity between patterns can be extrapolated to the general problem of multi-output learning.

We propose to solve these problem by using hierarchies of targets. In Chapter 3.4, we show a comprehensive summary of the methods and results of using this approach in the application of multi-fault prognosis.

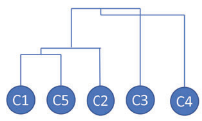
2.3.3 Extracting hierarchies with noise classes

When the extraction of hierarchies of classes is used for multi-class or multi-output problems there is always an implicit assumption: there are groups of classes that are more similar between each other than they are similar to others. In other words, there is an structure in how similar some classes are to each other.

But, what happens if we introduce classes that are equally similar to all the classes?

In Figure 2.14, we have an example of a hierarchy extracted with four classes. We use a hierarchical agglomerative clustering with average link to extract the hierarchy using the dissimilarity matrix. In total, there are 15 possible hierarchies for 4 classes. Let us assume that this is the hierarchy that results in the best performance when we train a hierarchical classifier on it and that any other hierarchy will result in a worse performance.

	C1	C2	C3	C4	C5
C1	0	0.55	0.9	0.9	0.5
C2	0.55	0	0.9	0.9	0.5
C3	0.9	0.9	0	0.8	0.5
C4	0.9	0.9	0.8	0	0.5
C5	0.5	0.5	0.5	0.5	0



MERGE	HEIGHT
C1 U C5	0.5
{C1,C5} U C2	0.525
{C1, C2, C5} U C3	0.77
{C1, C2, C3, C5} U C4	0.75

Figure 2.16: Hierarchy extracted for five classes using the dissimilarity metric. We use agglomerative clustering with average criterion.

If we add a new class, equally dissimilar to the rest, the process of extracting the hierarchy will be affected. Let us start by adding a class dissimilar from the rest, and easy to predict. In Figure 2.15, we have a depiction of what would happen. The structure of the hierarchy would be unchanged and the performance of the classifier trained on the hierarchy should perform similarly with respect to the four original classes.

On the other hand, in Figure 2.16, we illustrate what happens when we add a class that is equally dissimilar to the four original classes and noisy. The resulting hierarchy has changed, and the structure of the four original classes is not the same. As a result, the performance of the hierarchical classifier evaluated on the four original classes would change.

This toy example is a bit extreme, but it is not hard to imagine that in the presence of many "noise" classes, classes that are hard to predict and that are equally dissimilar to the rest of the classes, the process of extracting a good hierarchy becomes more difficult. As we add more and more noisy classes, the signal-to-noise ratio in the dissimilarity measures decreases and extracting hierarchies that result in a significant improvement for the classifiers trained on them becomes more and more challenging.

Chapter 3

Predictive Maintenance

Successful predictive maintenance solutions have been an important goal for many different industries. Not only does it promise optimal maintenance operations, an increment in productivity and decrease of operation costs; but it also prevents unexpected breakdowns that can have catastrophic consequences.

For many years, the standard has been to focus on particular components of the machine and design specific sensors and techniques to monitor its state and predict future faults. Which component to monitor is usually decided based on how critical it is for security (e.g., the brakes of a car), how frequently it has faults (e.g., rolling bearing mechanisms [8]), how mature is the technology, or a combination of all of the above.

With recent developments in sensor technologies, cloud connectivity or computing power; the field of predictive maintenance has had a surge in interest, investment and presence of products in many industries. Where we were using simple sensors to monitor faults [8], now we can use sophisticated cameras to detect defects on the equipment [6]; where we were training physics based models with low memory footprint [15], now we are deploying complex deep learning models [13].

But the advancement of technology has also given place for new opportunities. Now, we can go beyond the self-monitoring of selected components; we can focus on understanding the general behavior of the more complex systems and target components that are not necessarily critical for the operation of the machine. We can even study how different components in a complex machine interact and affect each other.

The seed of this thesis has been the PRIME project, whose goal was to predict faults in an already deployed fleet of sterilizers. Data had been recorded in these machines with the purpose of controlling the processes and securing the proper sterilization of the medical equipment.

The main characteristic of this setup, compared to the traditional single component cases discussed above, is the complexity of the machine in relationship with the recorded data: we want to predict and distinguish between many different types of faults with data describing the general behavior of the machine during real life operation. This presents challenges that we have summarized in research questions **AD-RQI** and **AD-RQII**: "how do we obtain useful information from repair logs for predictive maintenance?" and "can we use hierarchical approaches to improve fault detection and prognosis tasks?".

3.1 Realistic evaluation of models.

The goal of developing machine learning models for self-monitoring of complex machines is to improve maintenance operation, reducing costs and downtime of the machines. The end-user of this models, and the person that needs to interpret and exploit their outputs is the technician that will actually carry out the maintenance operation.

We aim to validate our models and proposed methodologies based on the utility and the improvement they provide to the end-user, not necessarily on the traditional machine learning performance metrics. In fact, we show that these two evaluation strategies might not always be aligned.

We realize this evaluation by understanding how the outputs of the self-monitoring models can be used in the field. In a simplified scenario, the output of the models will always be a suggestion of a maintenance operation. The final evaluation is then based on the costs and benefits of following these suggestions.

3.2 Learning predictive maintenance models using repair logs

3.2.1 Problem formulation

It is of great interest to learn predictive maintenance models from machines operating in the field. The behavior of a machine used in the field used by its users can be very different from how a machine is operated in controlled lab conditions. And it is precisely on the users' machines where it is more important to have successful predictive maintenance models.

Recording operational data describing the operation of the machines can be done automatically; it is relatively easy and cheap. However, collecting information detailing the failures that happen in the machines is much more costly and challenging. It first needs the diagnosis of a technician and the development of a system to identify, define, describe and delimit what types of faults happen in the machines. In the absence of these systems, the best source of information we can get is the sales logs.

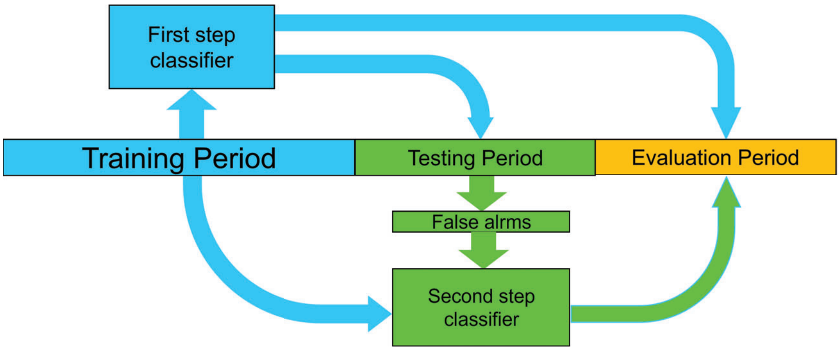


Figure 3.1: Workflow of the proposed methodology.

The information that we need is when and what broke in a machine. However, an entry in the sale log only means that a component was replaced; it does not mean that the replaced component was faulty. Furthermore, components can be replaced without the replacement being present in the logs.

3.2.2 Method

Our assumption is that if we can train a classifier on the data before a component replacement, and use that classifier to predict another component replacement accurately, then those two component replacements are most likely related to the same type of fault.

Our methodology is described in Figure 3.1. It consists of first selecting those component replacements that are successful at predicting each other. On its own, this first step would obviously tend to produce models that overfit the data. In the second step, we use the first step models and collect the false alarms. Due to the uncertainties in the labels, those false alarms can be actual false alarms or machines that actually are undergoing a fault. For each of these false alarms, we will extract a data sample with the sensor data immediately before the component replacement and add them to the selection process.

Our goal then is to select as many component replacement events as possible based on the ability to predict each other while keeping the number of the selected false alarms as low as possible. We achieve this by using a genetic algorithm using the fitness function presented in Algorithm 3.

3.2.3 Results

Data This experiment was carried out with data coming from sterilizers placed in different hospitals worldwide. The data was provided by our industrial partner, Getinge AB, in the framework of the PRIME project. Getinge also provides service and maintenance.

Algorithm 3 Fitness function for the refinement step.

N set of component replacement events and false alarm events.
$D_i = [(\mathbf{x^1}, y^1)...(\mathbf{x^m}, y^m)]$ dataset for event i
R vector of results
for $i = 1$ to $length(N)$ **do**
    $D_{train} = \bigcup_{j \neq i} D_j$
    $m =$ trainclassifier(D)
    $\mathbf{pred} =$ predictprobability(D_i)
    $R_i =$ measureAUC$(\mathbf{pred}, \mathbf{y_i})$
end for
F fitness value
for $r = 1$ to $length(N)$ **do**
    if i corresponds to a component event **then**
        if $R_i > threshold_1$ **then**
            $F = F + 1$
        else
            $F = F - 1/3$
        end if
    else
        if $R_i < threshold_2$ **then**
            $F = F + 1/10$
        end if
    end if
end for

We use two years of recorded data produced by the sterilizers. For each process of the machine (usually called cycle), the raw data contains sensor reading for the pressure and the temperature in different parts of the chamber. Working with the experts, we have extracted 86 features that characterize each cycle. For those two years of data, 275 maintenance events with component replacements have been recorded in the sales logs.

Evaluation The evaluation is based on the ability to predict a future component replacement. When the predictive model outputs an alarm, we simulate the situation where a technician is dispatched to the machine to check if the fault is happening or not. If a component replacement happens in the following 75 cycles, it is marked as a correct prediction. It is considered an early alarm if a component replacement happens between 75 and 100 cycles after the component replacement. If a component replacement is performed without a previous alarm, it is a missed failure. Finally, if no component replacement is performed in the next 100 cycles, it is a false alarm. In a realistic scenario, when an alarm is issued, the technician is sent to check the machine's status. If the technician finds that the machine is in a healthy state, we could consider the alarm to be false and the machine to be healthy for the near future. However, it is likely that the models would keep issuing alarms, which could be ignored based on the technician's expertise. For this reason, in our evaluation, we observe a cool down period of 25 cycles after a false alarm.

	CPRC	EA	MCR	FA
First Step	24	0	70	76
Second Step	30	0	64	67

Table 3.1: Results of predicting failures based on classifiers trained on period 1 (without filtering the false alarms) or periods 1+2 (including the component replacements from period 1 and the false alarms discovered in period 2). Comparison based on the Correctly Predicted Replaced Components (CPRC), Early Alarms (EA), Missed Component Replacement (MCR) and False Alarms (FA).

Result The comparison of the approach using only component replacement (first step) events and the approach adding the false alarms (second step) can be seen in Table 3.1. Not only is the Second Step reducing the number of false alarms by more than 10% (keep in mind that an unknown number of false alarms is to be expected); but we have also increased the number of correctly predicted component replacements by 25%.

There are many missed component replacements. As an indication, though, we should keep in mind that the genetic algorithm employed only selected 20-

25% of the component replacement events for training. This value roughly coincides with the ratio of correctly predicted component replacement to missed component replacements.

3.2.4 Conclusion

Our expectation was to reduce the number of false alarms after the refinement phase, which we have achieved by a margin of more than 10%, as verified experimentally on real-world industrial data.

However, more interestingly, we have also improved the number of correctly predicted component replacements by a healthy margin of 25% or more. Adding the false alarms to the training phase not only reduces the number of predicted false alarms in the future, but also improves the selection of component replacement events to create more accurate models.

3.3 Hierarchical classification for fault diagnostics

3.3.1 Problem formulation

In this section, we focus on the problem of fault diagnosis and the application of hierarchical classification. Fault diagnosis consists on, given a breakdown of a machine, discovering which specific component is responsible for the fault. To build the classification models, we use the sensor data describing the operation of machines.

In doing predictive maintenance tasks for complex machines, such as sterilizers or cars, we are faced with an scenario rarely tackled by the research community. The sensor data is not good enough to describe the state of each of the components that compose the machines. The main reason for this is that these complex machines have a very high number of components, and the data recorded is not recorded with the purpose of monitoring the state of each of the components. Usually, this data has been recorded for purposes such as control, security, or validation.

In addition, in many cases, individual faults are rare. In general, most of the components of the machine do not break during their lifetime, or if they break frequently, they might be subject to preventive maintenance operations. This means that for each individual component, the number of failures that we see can be scarce.

In this scenario, we assume that classification models aiming to isolate the responsible for a failure will have a low performance at the component level. However, we hypothesize that it is possible to build models able to classify the fault into a reduced group of components. This hypothesis is based on the intuition that different components, related through the tasks they perform in the machine, might cause the same symptoms in the data when they are faulty. For example, if we are measuring the engine's power output in a truck, and it is decreasing, there might be a fault happening in the engine or the injection systems, but not in the braking system.

To encode the different relationships of similarity between the machine's components, we propose using a hierarchy of components to train a multi-class classifier.

3.3.2 Method

Extracting a hierarchy of components To extract the hierarchy of components, we follow a top-down algorithm. Starting with all classes in the root node of the hierarchy, we iteratively split them into two subgroups until all classes are finally placed at the leaves.

To select the best possible split, we use a genetic algorithm that uses classifier-based distance as fitness function; we are looking for the split of components for which a classifier achieves the maximum performance.

Evaluation We measure the performance of the classifiers using traditional accuracy and the evaluation in terms of the realistic exploitation of the output of the model by a technician in a workshop.

The diagnostic model aims to assist the technician in performing the diagnosis on the machine, reducing the amount of work needed to find a fault. The output of a classification model is a numeric vector of probabilities, indicating how likely it is for each component to be the one responsible for the fault. The technician receives this input, selects the component with the highest probability, and performs a test to validate the predictions. If the component is not faulty, the technician will move to the next most likely one until he finally finds the fault.

The quality of any model in this framework can be evaluated based on its utility. We measure this utility by counting the number of tests that the technician needs to perform to identify the faulty component. Based on the ground truth, we can always know how many tests are necessary to find the real fault for every possible classification output. We will make three assumptions to simplify the experimental evaluation: all the tests consider a single component, the cost of every test is the same, and the tests are perfect.

3.3.3 Main Results

Data We use the data provided by one of our industrial partners, Volvo Trucks. We perform our analysis on a fleet of approximately 4000 trucks operating in Europe for three years. We use two sources of data: operational data and repair logs.

The operational data consists of onboard sensor readings, presented as timestamped instances with two weeks frequency. In total, we have 365 different numerical features collected from these sensors. In addition, we have 74 categorical features describing the particular configuration of each truck.

The repair logs contain information about the workshop visits of each truck. They specify which components have been replaced and when. It is important to notice that this information includes both replacements due to a breakdown and a scheduled maintenance operation.

Comparing utility and accuracy We select different sets of components to test the differences between a single classifier approach and our hierarchical approach. For each set, we select those components that have proven to be easier to identify.

	Accuracy		# of Tests	
# Comp	Single	Hierarchical	Single	Hierarchical
12	0.91	0.91	3.16	3.45
17	0.85	0.86	7.48	7.28
20	0.80	0.82	12	7.61
24	0.73	0.71	8.86	6.89
28	0.70	0.67	6.77	5.93
32	0.68	0.70	7.52	6.96
42	0.5	0.52	7.50	7.56

Table 3.2: Evaluation of a single classifier and our hierarchical approach with different sets of selected based on the accuracy and the number of tests it takes to isolate the fault.

In Table 3.2, we can see the comparison between the single classifier and the hierarchical classifier approaches in terms of accuracy. There does not exist a clear difference between the accuracies of the non-hierarchical and the hierarchical classifiers. As we increase the number of components, there is a clear drop in the accuracy. This is an expected result.

In the right side of Table 3.2, we can see the comparison of a single and a hierarchical classifier based on the number of tests needed to isolate the faulty component. With very few components, there does not exist a significant difference between the two approaches. However, as we add more components, the differences increase – in favor of the hierarchical approach. As we expected, as the number of components increases, so does the complexity of the relationship between them. Our hierarchical approach is able to encode these complex relationships and boost the performance.

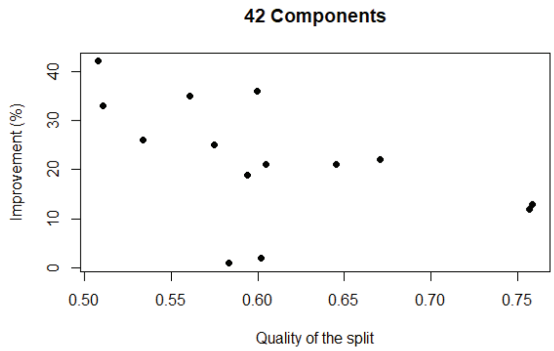


Figure 3.2: Relative improvement of the number of tests to isolate the selected components as a function of the quality of the split as measured in the hierarchy.

Nevertheless, as we continue to increase the number of components further, we see that this difference begins to decrease. By adding new components to our subset, we introduce components with lower predictive power and make the complete problem harder until the benefits of hierarchy disappear.

If we had evaluated our hierarchical approach only based on accuracy, we would have concluded that it does not provide any measurable improvement. However, in a more realistic scenario of interaction with the technician carrying out the diagnostics, we can measure significant benefits. Our hierarchical approach does not provide a better classification performance at the per-component level; however, by grouping the components at different levels, it allows us to create better models at these intermediate levels. It is precisely this structure in the predictions that can be exploited.

Enhancing the hierarchy If we want to enhance the overall performance of the diagnosis system, we need better data by, for example, adding new sensors to the machine. Encoded in the hierarchy is the information of similarity and dissimilarity between components based on their effect on the data. We can analyze this information to identify what kind of data is needed to obtain the biggest benefit.

In this experiment, we simulate the installation of a new sensor that helps us enhance the performance of the hierarchical model. It is not reasonable to assume that there exists a sensor that could distinguish between two random sets with an arbitrary number of components. However, we assume that installing a sensor that could distinguish between any two individual components is possible.

Using the hierarchy, we can identify which pair of components are harder to distinguish. We simulate this sensor by adding a feature highly correlated with the true class when it is one of the two components and random otherwise. For this experiment, we choose the case with 42 components. We choose the 14 nodes of the hierarchy where two components are distinguished. For each node, we simulate the addition of the new sensor and reevaluate the number of tests needed to isolate those two components. In Fig. 3.2, we can see the relationship between the relative improvement in the number of tests and the quality of the split as measured in the height of the hierarchy. The components at those lower nodes in the hierarchy present the larger benefit from adding the new sensor. However, the benefit is smaller for those nodes where the classifiers were already good enough.

3.4 Fault prognosis

3.4.1 Problem statement

The most valuable output of a predictive maintenance system is detecting a fault that will develop into a failure, identifying the fault, and predicting the precise when the failure will occur. With this information at hand, precise maintenance operations can be scheduled, reducing downtime, increasing the productivity of the machines, and reducing the cost of maintenance.

In complex systems with many different components and types of faults like a truck or a sterilizer, some faults show distinctive patterns in the data: distinct from the data produced by a healthy system and distinct from data associated with other types of faults. This is the ideal case, implicitly considered "the norm" and treated extensively in the literature; if it was the only case, one could simply focus on each type of fault individually. There are also other faults that do not affect the recorded data, and no approach is capable of predicting them. Finally, however, there are often sets of distinct faults that produce similar patterns in the recorded data.

Similarity is not a binary concept. There can be many different levels of similarity between faults in a complex machine with many different components and many different possible faults. We propose a new framework, HMP (Hierarchical Multi-fault Prognosis). HMP consists of first modeling these similarities by extracting a hierarchy of faults. Then, given such a hierarchy, for each internal node, we will train models to predict time-to-failures related to the faults in that node. The output of the node models is the estimated time-to-failure for any of the faults contained in the sub-tree expanded by the node.

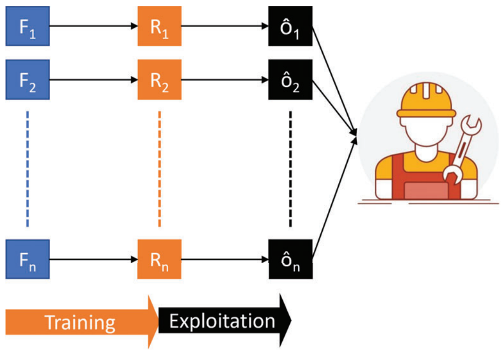


Figure 3.3: Flowchart for training individual models. For each type of fault F_i a regressor R_i is trained, that outputs an estimated time-to-failure $\hat{o}_i$. The list of the estimated times-to-failure is fed to the technician that will ultimately perform the maintenance if needed.

3.4.2 Method

Time-to-failure estimation for multivariate data We train our models using failure events for each type of fault. For each failure event, we obtain a dataset $D = (\boldsymbol{x}_t, T, f)$ containing, respectively: $\boldsymbol{x}_t$, the vector of values recorded in the machine describing its operation at time t; the corresponding measured time-to-failure, T; and an indicator of the fault happening in the machine, $f \in \{F_1, \ldots, F_n\}$, where F_i denotes each possible type of fault and n is the number of possible faults.

The general task is to train regressors to predict time-to-failure. In our case, the machines undergo only one fault at a time; however, we are interested in predicting, at any given moment, the time-to-failure for many different faults. Therefore, the output of our system is a vector of times-to-failure for the n possible faults:

$$R(\mathbf{x}) = [\hat{o}_1, \hat{o}_2, ...\hat{o}_n], \tag{3.1}$$

where R is our regressor system, and $\hat{o}_i$ is the estimated time to failure for component i.

Single-fault regressors to predict time-to-failure: contrast model The simplest approach to predict time-to-failure for n types of faults is to train n single-fault regressors. In Figure 3.3, we can see the workflow to train and use, or exploit, these models. A dataset is collected for each type of fault F_i, and individual regressors R_i are trained for each fault. Then, those models are used to predict time to failure for each type of fault $\hat{o}_i$, the final output presented to the technician.

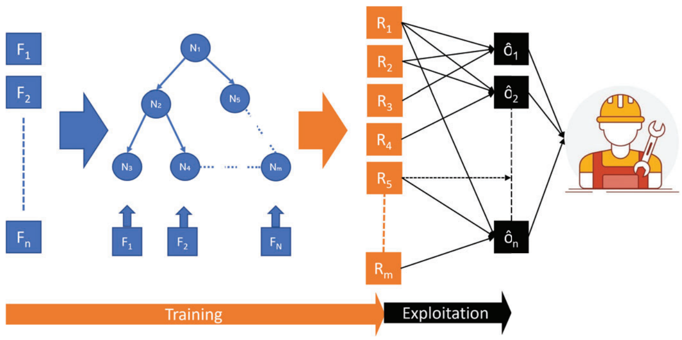


Figure 3.4: Flowchart for training hierarchical models. A hierarchy of faults is extracted from the data, then for each node N_i, a regressor is trained to predict time-to-failure for any of the faults contained in the node. The output of every regressor will be an estimated time-to-failure for all the faults contained in the node. The list of the estimated times-to-failure is fed to the technician that will ultimately perform the maintenance if needed.

In the contrast model approach, we train an individual model for each type of fault, but we also make use of the failure patterns associated with all the other faults, the contrast faults. The goal is to train a regressor that identifies the degree of evolution of the fault, discerning the faulty data from the healthy data, and at the same time can discern between different types of faults.

To achieve this, for the contrast faults' datasets, instead of using the true time-to-failure, we relabel them with a maximum time to failure T_{max}, effectively treating them as if it was data produced by a healthy machine. The training set we use for this models is therefore, $D_{F_i} = D(\boldsymbol{x}_t, T, f = F_i) \cup D(\boldsymbol{x}_t, T_{max}, f \neq F_i)$

HMP, Hierarchical Multi-fault Prognosis HMP groups the faults based on their similarity. If two faults are similar, as we have shown in Chapter 2.3.2, the contrast model will fail – but, intuitively, a regressor trained to predict *either* of the two faults would be much more successful. Since there can be many different degrees of similarities, we propose to extract a hierarchy of faults.

In Figure 3.4, we can see the workflow of the hierarchical approach. Using the data related to each of the faults, we extract a hierarchy of faults based on their similarities; then, at each node of the m modes, one regressor is trained to predict the time-to-failure for any of the faults contained in it. The output of each of the models is the estimated time-to-failure for any of the faults contained in this particular node. It is worth noting that, for a given type of fault i, many different nodes will be independently estimating its time-to-failure, based on different groupings.

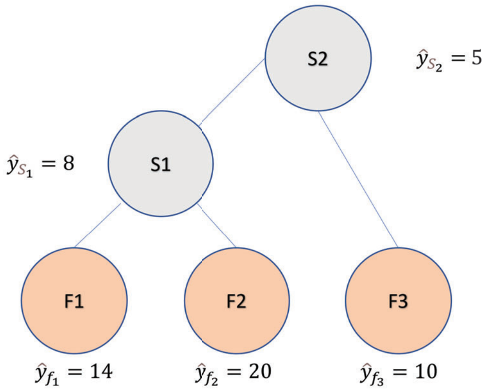


Figure 3.5: An example of a hierarchy of three faults: F1, F2 and F3; two super-types of faults: S1 and S2; and examples of predicted times to failure.

Our goal when extracting hierarchies is to group the most similar faults. This translates to first grouping faults with a high degree of similarity close to the bottom of the hierarchy, then those that are less similar, and finally those faults that are dissimilar close to the root of the tree.

There is not a single definition of similarity; in our case, we regard as similar those faults that produce similar failure patterns, $\boldsymbol{x}_t$. Using this definition, we characterize each fault by its failures patterns, i.e., the data recorded by the machine from the occurrence of the fault until its failure.

How to train a hierarchical regressor An example of a hierarchy is shown in Figure 3.5. The hierarchy consists of a set of nodes: the leaf nodes, each of them corresponding to one of the faults; and the internal nodes, each of them containing a set of faults.

We will use an approach similar to the flat contrast models. At each node, we will train a regressor using the failure patterns of the faults contained in the node labeled with their corresponding time to failure, and the rest of the faults' failure patterns as contrast. If S_n is the set of faults contained in the node and the S_c is the set of contrast faults (i.e., those not contained in the node), the training set will be $D_{node} = D(\boldsymbol{x}_t, T, f \in S_n) \cup D(\boldsymbol{x}_t, T_{max}, f \in S_c)$.

Using HMP to estimate time-to-failure Single-fault models were implicitly doing fault diagnosis, i.e., they were estimating time-to-failure for a specific fault. The output of an intermediate node is an estimated time-to-failure, but the fault responsible is one of the faults contained in the node.

If the estimated time-to-failure falls below a given threshold, an alarm is issued and a maintenance operation needs to be carried out to diagnose the fault and correct it if necessary. An alarm issued by a node regressor is in fact multiple alarms, one for each of the faults contained in the node. In order to rank the faults, and establish an order of preference for the technician, we will make use of the sub-tree expanded by the node.

The output of the prognostic system is a ranked list of estimated time-to-failures for the different faults. We rank the faults by going through the hierarchy starting from the selected node. To select among its children, we will pick the one that predicts the lowest time to failure until we reach one of the leafs.

Once we have found the most likely fault to be responsible for the predicted failure, we repeat the process omitting the selected fault. We repeat this procedure until all faults contained in the node are ranked. In the example in Figure 3.5, the lowest time-to-failure prediction corresponds to the $S2$ node, meaning that one of the faults contained in that node is predicted to be responsible for a failure in 5 units of time. The order of priority to check the different faults will be $F1, F2, F3$. Note that $F1$ and $F2$ are checked first because the predicted time to failure of $S2$ is smaller than the one of $F3$.

Evaluation In order to evaluate the predictive maintenance solution, we need to understand, and mimic, how this system will be used in practice. For each type of fault, the output of the predictive system is an estimated time-to-failure (sometimes called RUL, Remaining Useful Life). If we depend solely on these outputs to define our maintenance operation, we need to set a reasonable time threshold that gives enough time to schedule a maintenance operation before the failure happens.

The goal of the maintenance operation is to check the prediction and assess the state of the components related to the predicted fault. In practice, several alarms can happen simultaneously if the estimated times-to-failure for different faults all go below the set threshold. Therefore, a technician responsible for the maintenance should check the different alarms until the fault is found.

If the alarm points to a fault that is actually developing in the machine, the maintenance operation should avert the failure. If the alarms do not point at any fault developing at the machine, they will be false alarms. This simplified framework, gives us a straightforward method to evaluate the quality of our predictive models: we will quantify the percentage of Correctly Predicted Failures (CPF-rate), the percentage of False Alarms when the machine is Healthy (FAH-rate); the percentage of False alarms when a fault is happening, but the corresponding fault is Incorrectly Identified (FII-rate); and the number of tests that the technician needs to perform to isolate the responsible fault (NTest).

In order to ease the evaluation of the models, we will make some idealizations about the performance of the technician and the diagnostics tests that are carried out. The on-site tests are always correct: when a failure is predicted for a given fault, the diagnostics tests used to evaluate the health of the corresponding components are always correct.

Prediction models are never perfect. Sometimes, a model will continuously keep issuing alarms for the same fault for a long time. For the purpose of the evaluation, if a false alarm is issued and the technician deems it to be not happening, we assume that the fault is not happening and it will not lead to a failure in the near future, so the consequent false alarms will be dismissed.

Our motivating industrial scenarios share common characteristics regarding the costs of unexpected failures and false alarms. An unexpected breakdown is very costly; it means the interruption of service. The machine will not work again until a technician analyzes the machine, diagnoses, and corrects the problem, with all the possible extra delays including lack of necessary material for the repairs, testing the well-functioning of the machine, and so on.

A false alarm is costly since a technician needs to be dispatched to the machine (or the machine needs to be sent to the workshop) and must test the machine to check for the suggested fault. However, the operation of the machine is not affected if the maintenance is scheduled during a natural downtime of the machine. Therefore, the cost of a false alarm is significantly smaller than the cost of an unexpected failure.

In either of the situations mentioned above, more than one alarm can happen simultaneously. In those cases, the technician checks the possible faults iteratively, based on their estimated time-to-failure, until the fault is found. These extra tests on the machine result in an increased cost based on the time and resources needed. We assume that the costs related to checking for each of the simultaneous alarms are small compared to the overall cost of a maintenance intervention.

The specific scenario of the predictive maintenance of sterilizers and trucks is defined by: very complex machines, data that is not good enough to monitor every possible fault accurately, very high cost of unexpected breakdowns compared to false alarms. This scenario is common to many industrial situations.

3.4.3 Main Results

Data: The TEP Dataset The Tennessee Eastman Process (TEP) is a chemical process first introduced by [3]. The TEP consists of a reactor, a product cooler/condenser, a separator and a stripping column with the objective of reacting feed streams and separating the products. In simple terms, the TEP takes four input streams and converts the contents in the reactor into products.

There are different type of signals recorded, such as safety signals: values recorded to monitor some critical values which, if exceeded, would lead to hazardous situations in the plant; process signals, measuring the quantity and the quality of the products of the chemical plant; control variables, the operator or the control system manipulates these variables to control the operation of the plant; and monitor variables, not directly linked to the control or security.

Experimental setup To obtain our dataset, we sample 50 sequences of failure patterns for each of the 28 types of faults. First, we let the machine run for 1 hour without interference. Then, we introduce the fault, and we assume that the plant goes into failure and stops production after 2 hours of running with a fault. We record data for the 53 features every 3 minutes, i.e., we have sequences of 60 points, and the fault is injected after 20 measurements. We also extract sequences of data without the injection of fault to measure the FAH-rate, i.e., evaluating how often our models predict failure for completely healthy operation. All sequences of data are extracted independently, so we can safely use a 5-cross validation scheme.

Our objective is to learn models that predict the time to failure and identify the fault responsible for it. Without loss of generalization, we set a threshold of 45 minutes as the necessary time to execute maintenance before the failure. If our predicted time to failure goes below this number, an alarm is issued and an intervention by the technician is carried out.

We perform the model selection and final evaluation based on a cost analysis. We assume that the machines are running healthy most of the time, so the frequency of healthy state is 100 times higher than the faulty state. We assume that the cost of an undetected fault is 10 times higher than the cost of a visit of a technician. In addition, we assume that, once the machine is under maintenance, the cost of each test adds 0.1 times more cost to the maintenance operation.

Results We are going to evaluate the models created using our framework HMP. We are going to compare different approaches to extract the hierarchies and benchmark it against the single-fault approaches. Our first step is to evaluate our approach on a randomly generated hierarchy. Comparing against a random hierarchy gives us an adequate benchmark to validate the method to extract hierarchy. At each fold, we will draw a new hierarchy, train, evaluate the models for each node, and select the combination of models that provide useful results, similar to how we did in the previous subsection. The results are detailed in the second row of Table 3.3.

There exists a slight difference in the number of correctly predicted faults between the random hierarchy and the single-fault contrast model, and as a consequence, a difference in the final cost. However, this difference does not prove to be statistically significant after applying the Student's t-test.

In Table 3.3, we can also observe the detailed performance of the models trained on hierarchies using six different extraction methods. We extract hierarchies using the classifier based distance (CBD), using area under the ROC curve as performance, and using the euclidean distance in the feature space

	CPF	FAH	FII	# of tests	Final cost
Contrast	0.50±0.03	0±0	0±0	1	0.50 ±0.03
Random	0.52±0.06	0±0	0.003 ±0.003	1.17 ±0.18	0.48 ±0.06
CBD K-means	0.59±0.01	0±0	0.004±0.003	1.24 ±0.01	0.41 ±0.01
CBD Bal K-means	0.58±0.02	0±0	0.003±0.002	1.19 ±0.08	0.42 ±0.02
CBD Agglo	0.55±0.06	0±0	0.002±0.002	1.3 ±0.01	0.45 ±0.05
Centroid k-means	0.55±0.003	0 ±0	0.003±0.004	1.22 ±0.28	0.46 ±0.03
Centroid Bal k-means	0.54±0.006	0±0	0.005±0.002	1.17 ±0.08	0.46 ±0.06
Centroid Agglo	0.53±0.004	0±0	0.003±0.005	1.18 ±0.23	0.47 ±0.04

Table 3.3: Comparison of HMP with different methods to extract the hierarchies.

between the centroids of the different types of faults' failure patterns. We also use three different clustering algorithms: agglomerative clustering, divisive k-means, and divisive balanced k-means.

The best performance is achieved by the combination of CBD and K-means. Up to 60% of the failures are correctly predicted, keeping the number of false alarms close to 0. There exists a statistically significant difference between this approach and the flat models with contrast. The improvement over the single-fault model with contrast originates from the models created at the internal nodes of the hierarchy. At least some of them contain faults whose failure patterns are sufficiently similar.

There exists also a statistically significant improvement over HMP using the random hierarchy. This means that in the data, there exist complex and distinctive relationships of similarity between the different failure patterns, complex and precise enough that they can not be found randomly.

On the other hand, hierarchical approaches need a (slightly) higher number of tests to find the responsible fault. This is to be expected. It is our assumption that in the presence of the similar faults, the contrast approach will fail to correctly predict the time-to-failure. A hierarchical approach, able to identify these similar faults and cluster them together, should create a model able to correctly estimate the time-to-failure to one of the similar faults; but it would be less accurate in selecting exactly which of the faults is the responsible one. In our scheme of costs, where the effort associated with the extra tests needed to isolate the fault is low compared to the cost of false alarms and much lower than the cost of an unpredicted failure, it is clearly acceptable to have this increase in the number of tests in exchange of correctly predicting more faults.

CBD based hierarchies in general outperform hierarchies that used euclidean distance between centroids. Centroid based hierarchies seem to improve

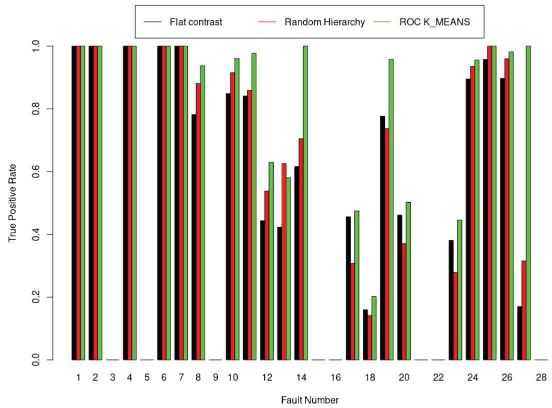


Figure 3.6: CFP per class evaluated for three different approaches: contrast model, random hierarchy, and the hierarchy obtained combining the CBD dissimilarity and the k-means divisive clustering algorithm.

the flat model with contrast and the random hierarchy, but the difference is not significant. Euclidean distance can suffer from the curse of dimensionality, specifically, in our case, difference between failure patterns can appear in few specific features, while the rest of the features, which contain a degree of noise in them, remain unaffected by the fault, rendering the calculation of the distance between centroids irrelevant.

Analyzing the results of the different clustering algorithms, we can observe that divisive k-means seems to outperform, marginally, the divisive balanced k-means, and this in turn outperform marginally the agglomerative clustering. None of these differences are significant. We can conclude that, for this specific task and dataset, the biggest source of improvement comes from the method to measure the similarities between types of faults. Once the similarities have been computed, the different clustering algorithms to extract the hierarchies seem to provide marginal gains.

In Figure 3.6, we can see depicted the difference in percentage of faults predicted (CFP) for each type of fault for the individual contrast model, the HMP trained on a random hierarchy and the HMP trained on the best hierarchy. There are three sets of faults, based on the results of the different approaches. There are faults whose failures seem impossible to predict, those are faults 3, 5, 9, 15, 16, 21, 22 and 28. There are failures whose faults are "easy" to predict, all approaches achieve maximal performance: 1, 2, 4, 6 and 7. Finally, we have the "interesting" faults, those types of faults for which some failures can be predicted and some failures can not be predicted depending on the approach taken. Those are faults 8, 10, 11, 12, 15, 17, 18, 19, 20, 23, 24, 25 and 26. If we compare the performances of the flat models with contrast and the model using a random hierarchy, we can see that the random hierarchy tends to outperform the flat models for the "interesting" faults, although there are counterexamples.

The hierarchy using the area under the ROC curve and using a divisive clustering algorithm using k-means always (except one case in fault 13) outperforms the other two approaches. The improvements are not very high for most faults, except for classes 14, 19 and especially 27.

In a complex system, with many types of faults, there are going to be many faults that can be easily predicted, many faults that can not be predicted at all, and some faults where improvements can be achieved by making the right decisions. In our case, evaluating the performance of the whole system, predicting the 28 faults, one can conclude that the hierarchical approach provides a statistically significant but humble improvement over the contrast models. However, if we analyze the results on a per-fault basis, we can see dramatic improvements, for example, for fault 27 we move from less than a 20% CFP to 100%.

Chapter 4

Summary of papers

This section introduces the motivation, objectives and results of each individual paper and how they relate to the overall objectives of this thesis.

4.1 Paper I - Filtering Misleading Repair Log Labels to Improve Predictive Maintenance Models

One of the main challenges for predictive maintenance in real applications is the quality of the data, especially the labels. In this paper, we proposed a methodology to filter out the misleading labels that harm the performance of Machine Learning models. Ideally, predictive maintenance would be based on the information of when a fault has occurred in a machine, and what specific type of fault it was. Then, we could train machine learning models to identify the symptoms of such fault before it leads to a breakdown. However, in many industrial applications, this information is not available. Instead, we approximate it using a log of component replacements, usually coming from the sales or maintenance departments. The repair history provides reliable labels for fault prediction models only if the replaced component was indeed faulty, with symptoms captured by collected data, and it was going to lead to a breakdown.

However, very often, at least for complex equipment, this assumption does not hold. Models trained using unreliable labels will then, necessarily, fail. We demonstrate that filtering misleading labels leads to improved results. Our central claim is that the same fault, happening several times, should have similar symptoms in the data; thus, we can train a model to predict them. On the contrary, replacements of the same component that do not exhibit similar symptoms will be confusing and harm the ML models. Therefore, we aim to filter the maintenance operations, keeping only those that can be used

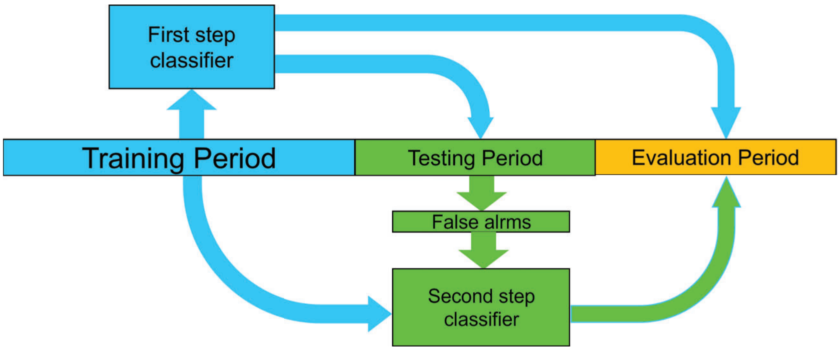


Figure 4.1: Workflow of the methodology proposed in Paper I.

to predict each other. If we can train a successful model using the data before a component replacement to predict another component replacement, then those maintenance operations must be motivated by the same, or a very similar, type of fault.

On its own, a filtering approach based on this approach will inevitably result in an overfitted model, i.e., a high number of false alarms. Our full approach, exemplified in Figure 4.1, uses two steps to filter the repair logs. On a first step, we use a genetic algorithm to select those maintenance repairs that can predict each other and train a predictive model. On the second period, we evaluate the model created in the first period and compute the false alarms. The final selection is achieved with another genetic algorithm to maximize the number of maintenance operations that can predict each other while keeping the number of false alarms as low as possible.

Author Contributions: The author of this thesis designed and implemented the work presented in the paper, performed the experiments, wrote most of the text of the paper, and created all visualizations.

4.2 Paper II - Hierarchical Multi-class Classification for Fault Diagnosis

This paper formulates the problem of predictive maintenance for complex systems as a hierarchical multi-class classification task. This formulation is useful for equipment with multiple sub-systems and components performing heterogeneous tasks. Often, the data available describes the whole system's operation and is not ideal for accurate condition monitoring. In this setup, specialized predictive models analyzing one component at a time rarely perform much better than random. However, using machine learning and hierarchical approaches, we can still exploit the data to build a fault isolation system that provides measurable benefits for technicians in the field.

We propose a method for creating a taxonomy of components to train hierarchical classifiers that aim to identify the faulty component. The output of this model is a structured set of predictions with different probabilities for each component.

Using a dataset from a real-case problem coming from the automotive industry, we demonstrate how traditional machine learning performance metrics, like accuracy, fail to capture practical benefits. Our proposed hierarchical approach succeeds in exploiting the information in the data and outperforms non-hierarchical machine learning solutions. In addition, using hierarchies, we can understand which components' faults are harder to predict, allowing us to identify which sensors can be added to improve the overall predictive system

Author Contributions: The author of this thesis designed and implemented the work presented in the paper, performed the experiments, wrote most of the text of the paper, and created all visualizations.

4.3 Paper III - Hierarchical multi-fault prognostics for complex systems

The field of predictive maintenance for complex machinery with multiple possible faults is an important but largely unexplored area. In general, one assumes, often implicitly, the existence of monitoring data specific enough to capture every possible fault independently from all the others.

In this paper, we focus on the problem of predicting time-to-failure, or remaining useful life, in situations where the above assumption does not hold. Specifically, what happens when the data is not good enough to uniquely predict every fault, and, more importantly, what happens when different faults share the same symptoms on the recorded data.

We demonstrate that prognostics approaches learning independent models for each fault are inadequate. In particular, in the presence of faults that produce similar failure patterns, they produce false alarms disproportionately often or miss the majority of failures.

We propose the HMP framework (Hierarchical Multi-fault Prognosis) to solve this problem by extracting a hierarchy of faults based on the similarity of the data they produce. At each node of the hierarchy, we train a regression model to predict the time-to-failure for any of the faults contained in this node. The intuition is that while it might be impossible to predict individual time-to-failure in the presence of similar faults, a model trained on aggregated data can still provide useful information. We demonstrate through experiments the validity of our approach.

Author Contributions: The author of this thesis designed and implemented the work presented in the paper, performed the experiments, wrote most of the text of the paper, and created all visualizations.

4.4 Paper IV - Pitfalls of Assessing Extracted Hierarchies for Multi-Class Classification

Using hierarchies of classes is one of the standard methods to solve multi-class classification problems. In the literature, selecting the right hierarchy is considered to play a key role in improving classification performance. Although different methods have been proposed, there is still a lack of understanding of what makes a hierarchy good and what makes a method to extract hierarchies perform better or worse.

To this effect, we analyze and compare some of the most popular approaches to extracting hierarchies. We identify some common pitfalls that may lead practitioners to make misleading conclusions about their methods. In addition, to address some of these problems, we demonstrate that using random hierarchies is an appropriate benchmark to assess how the hierarchy's quality affects the classification performance.

In particular, we show how the hierarchy's quality can become irrelevant depending on the experimental setup: when using powerful enough classifiers, the final performance is not affected by the quality of the hierarchy.

Our results confirm that datasets with a high number of classes generally present complex structures in how these classes relate to each other. In these datasets, the right hierarchy can dramatically improve classification performance.

Author Contributions: The author of this thesis designed and implemented the work presented in the paper, performed the experiments, wrote most of the text of the paper, and created all visualizations.

4.5 Paper V - Why is Multi-Class Classification Hard?

In classification problems, as the number of classes increases, correctly classifying a new instance into one of them is assumed to be more challenging than making the same decision in the presence of fewer classes. The essence of the problem is that using the learning algorithm on each decision boundary individually is better than using the same learning algorithm on several of them simultaneously. However, why and when it happens is still not well-understood today. This work's main contribution is to introduce the concept of heterogeneity of decision boundaries as an explanation of this phenomenon.

Based on the definition of heterogeneity of decision boundaries, we analyze and explain the differences in the performance of state of the art approaches to solve multi-class classification. We demonstrate that as the heterogeneity increases, the performances of all approaches, except one-vs-one, decrease. We show that by correctly encoding the knowledge of the heterogeneity of decision boundaries in a decomposition of the multi-class problem, we can obtain better results than state of the art decompositions. The benefits can be an increase in classification performance or a decrease in the time it takes to train and evaluate the models.

We first provide intuitions and illustrate the effects of the heterogeneity of decision boundaries using synthetic datasets and a simplistic classifier. Then, we demonstrate how a real dataset exhibits these same principles, also under realistic learning algorithms. In this setting, we devise a method to quantify the heterogeneity of different decision boundaries, and use it to decompose the multi-class problem. The results show significant improvements over state-of-the-art decompositions that do not take the heterogeneity of decision boundaries into account.

Author Contributions: The author of this thesis designed and implemented the work presented in the paper, performed the experiments, wrote most of the text of the paper, and created all visualizations.

Chapter 5

Conclusion

In this chapter, we conclude this thesis by first revisiting research questions and providing a summary of the main takeaway messages and the major contributions of the work. Then we present interesting future research directions that could stem from the work presented here.

5.1 Revisiting The Research Questions

This thesis revolves around the concept of extracting hierarchies of classes from the data and the utilization of these hierarchies to improve the performance of supervised learning tasks. We have applied these techniques in the application domain of self-monitoring of complex machines.

Carrying out diagnostics or prognostics using real-life data coming from complex machines is a generally overlooked problem in the scientific literature. From the exploitation of information from repair logs to do predictive maintenance, to the developing of new techniques to deal with the inadequacy of the data to monitor each and every single component requires for new technical and theoretical innovations.

Our application domain has served as an inspiration, motivation, and test field for our theoretical contributions, but it has also allowed us to prove the utility of our approaches beyond the contributions to the state-of-the-art of machine learning and data science.

Here, we review the research questions presented in Section 1.3 based on the contributions and findings presented in different works in this thesis.

Application Domain Research Question I: How do we obtain useful information from repair logs for predictive maintenance? In Paper I, we have developed a novel filtering approach to solve the problem of the inaccuracies stemming from using repair logs as a source of labeling for sensor data. This approach has proven to, as expected, reduce the number of false alarms that are produced by just selecting those component replacements that are good at predicting each other. Furthermore, it has also succeeded in increasing the ability to predict more faults.

Application Domain Research Question II: Can we use hierarchical approaches to improve fault detection and prognosis tasks? In Paper II, we have used hierarchical multi-class classification for fault diagnosis. Our proposed hierarchical approach does not show significant improvement over non-hierarchical approaches in terms of traditional machine learning performance metrics; however, it does show improvement when measuring the utility that a fault diagnosis system based on its predictions would get. Furthermore, we have shown that the hierarchy of faults extracted can be used to direct the future improvement of the diagnostic system, by isolating those faults that are more similar and would benefit more from an investment in developing better monitoring equipment to diagnose their faults.

In Paper III, we have used a hierarchy of faults to perform multi-output prognosis. Hierarchical multi-output regression is a very untreated topic in the literature, and to the best of our knowledge has not been used in the field of predictive maintenance before. We have shown how multi-fault prognostic is particularly sensitive to the presence of different faults that produce similar symptoms in the data, reducing the performance of the learned models dramatically. Our proposed method is able to overcome this problem by grouping similar faults together and learning regression models able to predict the time-to-failure of any of those faults.

Theoretical Research Question I: How do we develop methods to extract hierarchies from data? In Paper IV, we have dealt with common pitfalls found in the literature about the evaluation of methods to extract hierarchies from the data. Traditionally, contributions in the topics of hierarchy extraction for multi-class classification tend to miss a rigorous evaluation of the extraction method. We propose using random hierarchies as an appropriate benchmark to be able to isolate the quality of the extracted hierarchy, isolating its effects from other aspects of the problem.

Evaluating the quality of the extracted hierarchy through the performance of a hierarchical classifier trained on it is also problematic. We have proven that for a powerful enough classifier, the quality of the hierarchy becomes irrelevant.

Finally, we have also studied the utilization of classifier based distance as a way to measure dissimilarity between classes to extract hierarchies. It is generally assumed that classifier based distance is going to be a good dissimilarity, but this will intuitively depend highly on what type of learning algorithm is used to measure the dissimilarity, but less intuitively on the learning algorithm used for the hierarchical classifier.

In Paper V, we have proposed a new dissimilarity measure based on the heterogeneity of decision boundaries.

Theoretical Research Question II: What makes some hierarchies better than others? In Paper V, we have introduced and extensively discussed the concept of heterogeneity of decision boundaries. The heterogeneity of decision boundaries provides, at least partially, an explanation as to why some decomposition methods work better than others. We have analyzed how the heterogeneity of decision boundaries affects traditional methods to decompose the multi-class classification into smaller problems, and how it explains the differences in performance between them. We have also shown that approaches that decompose the original multi-class problem taking into account the heterogeneity present significant improvements compared with approaches that do not take it into account.

In Paper III, we have shown how the similarity between patterns can dramatically affect the performance of multi-output regression problems.

In Paper II, we have briefly discussed the problem of "noise" classes in the process of extracting hierarchies. The intuition behind the extraction of hierarchies is that there exists a group of classes more similar between them, than they are similar to the rest of the classes, i.e., that there exist structure in these relationships of similarity. However, in the presence of noisy classes, equally dissimilar to all the classes, the similarity of structure can be submerged in the noise, and the extraction can become impossible.

5.2 Future Work

In this thesis and the attached papers, we have emphasized evaluating the quality of different approaches based on their utility for the final user. In our case, we have used simplistic simulated scenarios, where we have counted the number of false alarms, correctly predicted failures, or the number of tests it takes to isolate a fault.

A fundamental question yet to be answered in the scientific literature is how to implement these models in real-life scenarios and convince the final users of their utility. Maintenance of complex equipment such as cars, trucks, or sterilizers is carried out by expert technicians able to recognize, identify, and correct faults. How to integrate their expertise with predictive models using sensor information and sophisticated signal processing and machine learning techniques can be very challenging, especially since these technicians are experts in identifying and diagnosing faults when breakdowns have already happened. The symptoms of a very advanced fault can be very different from the early symptoms of an emerging fault.

We believe that the organization of faults in hierarchical structures can be instrumental in this process. Furthermore, the process of extracting hierarchies can benefit from the technicians' expertise. For example, faults can be grouped based on the function of the different components they are related to.

We have evaluated our hierarchies based on the final performance at the leaf level and the performance at identifying individual faults. One of the assumptions we made in papers II and III is that the technician had to perform one test per suggested component to identify faults and that all tests had similar costs. This process is more complex in real life. There can be simple and cheap tests to discern between many faults (e.g., if a car does not start up, the fault is probably located in the battery system and definitely not in the braking system) and very complex and costly tests to check for other individual faults.

Hierarchies of faults allow us to develop global predictive maintenance operations taking these heterogeneities into account, allowing for hybrid solutions where both expert knowledge and machine learning models can coexist. Based on their respective performance, at some levels of the hierarchy, the decision can be made by an expert running one of these simple tests, whereas, at other levels of the hierarchy, we could rely on machine learning models to identify and evaluate complex symptoms.

We have explored in depth how different methods of extracting hierarchies affect the performance of multi-class classifiers. We have also shown that in multi-output regression, there is a measurable improvement by using "informed" hierarchies over random ones. However, there is still a gap in understanding how different hierarchies and different methods to extract hierarchies can affect the multi-output and multi-label cases.

In hierarchical multi-class classification, we are making an "OR" decision at every node: either the left or right child node is selected. However, this decision becomes "OR/AND" in the multi-label or the multi-class case. Given that there is experimental evidence that there are better and worse hierarchies, it is interesting to explore what characteristics of the data influence these differences and find which are the best methods to extract hierarchies for these tasks.

References

[1] Mohamed Aly. Survey on multiclass classification methods. *Neural Netw*, 19, 2005. (Cited on page 13.)

[2] Lior Bachar, Renata Klein, Moshe Tur, and Jacob Bortman. Fault diagnosis of gear transmissions via optic fiber bragg grating strain sensors. *Mechanical Systems and Signal Processing*, 169, 2022. (Cited on page 3.)

[3] J.J. Downs and E.F. Vogel. A plant-wide industrial process control problem. *Computers and Chemical Engineering*, 17, 1993. Industrial challenge problems in process control. (Cited on page 49.)

[4] Idriss El-Thalji and Erkki Jantunen. A summary of fault modelling and predictive health monitoring of rolling element bearings. *Mechanical Systems and Signal Processing*, 60-61, 2015. (Cited on page 3.)

[5] Shantanu Godbole. Exploiting confusion matrices for automatic generation of topic hierarchies and scaling up multi-way classifiers. *Progress Rep., IIT Bombay*, 2002. (Cited on pages 15 and 22.)

[6] Mehmet Karakose and Orhan Yaman. Complex fuzzy system based predictive maintenance approach in railways. *IEEE Transactions on Industrial Informatics*, 16, 2020. (Cited on page 35.)

[7] Tao Li, Shenghuo Zhu, and Mitsunori Ogihara. Hierarchical document classification using automatically generated hierarchy. *J. Intell. Inf. Syst.*, 29, 2007. (Cited on page 15.)

[8] Xinsheng Lou and Kenneth A Loparo. Bearing fault diagnosis based on wavelet transform and fuzzy inference. *Mechanical Systems and Signal Processing*, 18, 2004. (Cited on page 35.)

[9] Yaqiong Lv, Wenqin Zhao, Zhiyao Zhao, Weidong Li, and Kam K.H. Ng. Vibration signal-based early fault prognosis: Status quo and applications. *Advanced Engineering Informatics*, 52, 2022. (Cited on page 4.)

[10] S.A. McInerny and Y. Dai. Basic vibration signal processing for bearing fault detection. *IEEE Transactions on Education*, 46(1), 2003. (Cited on page 3.)

[11] Vanja Mileski, Sašo Džeroski, and Dragi Kocev. Predictive clustering trees for hierarchical multi-target regression. In *Advances in Intelligent Data Analysis XVI*, 2017. (Cited on page 13.)

[12] Dubravko Miljković. Fault detection methods: A literature survey. In *2011 Proceedings of the 34th International Convention MIPRO*, 2011. (Cited on page 4.)

[13] Khanh T.P. Nguyen and Kamal Medjaher. A new dynamic predictive maintenance framework using deep learning for failure prognostics. *Reliability Engineering and System Safety*, 188, 2019. (Cited on page 35.)

[14] Yannis Papanikolaou, Grigorios Tsoumakas, and Ioannis Katakis. Hierarchical partitioning of the output space in multi-label data. *Data and Knowledge Engineering*, 116, 2018. (Cited on page 13.)

[15] K.S. Park. Condition-based predictive maintenance by multiple logistic function. *IEEE Transactions on Reliability*, 42, 1993. (Cited on page 35.)

[16] Yashoteja Prabhu, Anil Kag, Shrutendra Harsola, Rahul Agrawal, and Manik Varma. Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In *Proceedings of the 2018 World Wide Web Conference*, 2018. (Cited on page 13.)

[17] Chandan K Reddy and Bhanukiran Vinzamuri. A survey of partitional and hierarchical clustering algorithms. In *Data clustering*. 2018. (Cited on page 15.)

[18] Carlos N Silla and Alex A Freitas. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22(1), 2011. (Cited on page 13.)

[19] Daniel Silva-Palacios, Cèsar Ferri, and María José Ramírez-Quintana. Improving performance of multiclass classification by inducing class hierarchies. *Procedia Computer Science*, 108, 2017. (Cited on pages 13 and 22.)

[20] Yingying Zhao, Dongsheng Li, Ao Dong, Dahai Kang, Qin Lv, and Li Shang. Fault prediction and diagnosis of wind turbine generators using scada data. *Energies*, 10, 2017. (Cited on page 4.)