Knowledge-Enhanced Machine Learning for Diagnosis

Louis Goupil

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

l'Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)

Présentée et soutenue le *08/07/2024* **par :**

Louis Goupil

Knowledge-Enhanced Machine Learning for Diagnosis

JURY

Jean-Marie Lagniez	Professeur des Universités	Président du Jury
Vincent Cocquempot	Professeur des Universités	Membre du Jury
Alexandre Voisin	Maître de Conférences	Membre du Jury
Louise Travé-Massuyès	Directrice de Recherche	Directrice de thèse
Elodie Chanthery	Maîtresse de Conférences	Co-directrice de thèse
Sébastien Delautier	Ingénieur Recherche	Invité
Hubert Lespinasse	Ingénieur Recherche	Invité

École doctorale et spécialité :

EDSYS : Informatique 4200018

Unités de Recherche :

l'équipe DISCO du LAAS-CNRS et l'Inno'lab Toulouse d'Atos

Directrices de Thèse :

Louise Travé-Massuyès et *Elodie Chanthery*

Rapporteurs :

Vincent Cocquempot et *Alexandre Voisin*

INSTITUT NATIONAL DES SCIENCES APPLIQUÉES DE TOULOUSE

Abstract

LAAS-CNRS - Atos

Doctor of Philosophy

Knowledge-Enhanced Machine Learning for Diagnosis

by Louis Goupil

Model-based diagnosis requires full knowledge about the analyzed system. On the other hand, data-driven diagnosis lacks explanations about the cause of a fault. Many applications require reliable and explainable fault diagnosis and would benefit from knowing the cause of faults in order to avoid them. In this thesis, the focus is on developing new methods combining model-based and data-driven diagnosis in a synergistic way. Specifically, the emphasis is on structural analysis as a model-based method that requires only knowledge of the system's structure. For that purpose, a novel explainable method has been designed called DT4X (Diagnosis Tree for eXplainability). It leverages decision trees where decisions are informed by diagnosis meta knowledge, specifically focusing on the properties of diagnosis indicators. This knowledge is used at each node to articulate a symbolic classification problem, outputting discriminating functions. The outcome is a multivariate decision tree that produces a compact model for diagnosis. The use of decision trees increases the explainability of the outcome, all the more so as one discovers the explicit formal expressions of diagnosis indicators, structured in the form of analytical redundancy relations. On simple systems, DT4X proves to output expressions that could previously only be found with full physical knowledge of the system. Its accuracy is higher than traditional machine learning algorithms. On more complex dynamic systems, DT4X reaches very high accuracy but lacks interpretable insight about the studied system. On logical circuits, a preprocessing of the data is proposed to remove samples corresponding to masked faults. DT4X finds logical expressions that possess all the properties of model-based diagnosis indicators. A variant of DT4X has been developed called PI-DT4X (Physics Informed DT4X). It is an alternative that requires more physical insight about the system but has higher accuracy and capacity to find relevant diagnosis indicators. PI-DT4X takes as input the structural model of the system and injects specific structural sub-models in the decision tree to guide and focus symbolic regression, so that diagnosis indicators are discovered faster and easier.

Keywords: Model-based and Data-based Diagnosis, Machine Learning, Hybrid Artificial Intelligence, Business Knowledge, Additive Manufacturing.

Le diagnostic basé modèle requiert une connaissance complète du système analysé. Par ailleurs, le diagnostic basé sur des données ne fournit pas d'explications sur la cause d'une défaillance. De nombreuses applications nécessitent un diagnostic des défauts fiable et explicatif, et bénéficieraient d'obtenir la cause des défaillances afin de les éviter. Dans cette thèse, l'accent est mis sur le développement de nouvelles méthodes combinant le diagnostic basé modèle et le diagnostic basé données de manière synergique. Plus précisément, l'accent est mis sur l'analyse structurelle en tant que méthode basée modèle qui ne nécessite que la connaissance de la structure du système. Dans ce but, une nouvelle méthode fournissant des explications a été conçue, appelée DT4X (Diagnosis Tree for eXplainability). Elle exploite les arbres de décision où les critères de décision sont construits à partir de méta-connaissance des méthodes de diagnostic basés modèle, se concentrant spécifiquement sur les propriétés des indicateurs de diagnostic. Cette connaissance est utilisée à chaque nœud pour articuler un problème de classification symbolique, produisant des fonctions discriminantes. Le résultat est un arbre de décision multivarié qui produit un algorithme de diagnostic. L'utilisation d'arbres de décision augmente l'explicabilité du résultat, d'autant plus que l'on découvre les expressions formelles explicites des indicateurs de diagnostic, structurées sous forme de relations de redondance analytique. Sur des systèmes simples, DT4X s'avère produire des expressions qui ne pouvaient auparavant être trouvées qu'avec une connaissance physique complète du système. Sa précision est supérieure à celle des algorithmes d'apprentissage automatique traditionnels. Sur des systèmes dynamiques plus complexes, DT4X atteint une précision très élevée mais apporte peu d'explications sur le système étudié. Sur les circuits logiques, un prétraitement des données est proposé pour supprimer les échantillons correspondant à des défaillances masquées. DT4X trouve des expressions logiques qui possèdent toutes les propriétés des indicateurs de diagnostic basés modèle. Une variante de DT4X a été développée, appelée PI-DT4X (Physically Informed Diagnosis Tree for eXplainability). Il s'agit d'une alternative qui nécessite une meilleure compréhension physique du système mais présente une précision plus élevée et une meilleure capacité pour trouver des indicateurs de diagnostic pertinents. PI-DT4X prend en entrée le modèle structurel du système et injecte des sous-modèles structurels spécifiques dans l'arbre de décision pour guider et concentrer la régression symbolique, de sorte que les indicateurs de diagnostic soient découverts plus rapidement et plus efficacement.

Mots-clés : Diagnostic basé modèle et basé données, Apprentissage Machine, Intelligence Artificielle Hybride, Connaissances Métier, Fabrication Additive.

Acknowledgements

First, I wish to thank Vincent Cocquempot, Alexandre Voisin and Jean-Marie Lagniez for agreeing to evaluate my work and taking the time to attend the defense in person. Thank you very much for your kind, relevant and constructive questions and remarks, both in the reports and at the defense. They allowed me to improve and finalize this manuscript.

I want to express my deepest gratitude to my three supervisors, Elodie Chanthery, Louise Travé-Massuyès and Sébastien Delautier. Thank you for always being available, caring, listening and open-minded. Thank you for, at times, going beyond your obligations as thesis supervisors. Thank you for creating an environment in which I can grow and feel at ease. It has always been a pleasure for me to work and interact with you, and I consider that paramount. In particular, thank you Sébastien for your guidance and patience. Thank you also for giving me the opportunity to have a first teaching experience. I know I am not the only one thinking that you were a good team leader. Louise and Elodie, thank you for always finding the right way to talk to me, encourage me, and push me to go beyond what I thought were my limitations. I will be forever grateful for all the things you taught me during these three years and all that I got to experience.

I want to thank all the members of the Inno'lab Toulouse of Atos. Through these three years I had the chance to work alongside many different persons including Margot, Étienne, Léo, Clara, Darsana, Clément, Maxence. Thank you all for the great atmosphere. I also was lucky enough to have many colleagues helping with my research topic, including Julia, Dorian, Thibault, Pauline and Sonia. I am grateful for your great contributions to the project, this would not have been possible without you. In particular, I want to express my thanks to Hubert Lespinasse for supporting me throughout the three years with advices of all kinds, technical support and for attending the defense as an Atos representative. In addition, thank you very much Audrey for supervising the publication of two patents and the valuable friendship we built.

Similarly, I want to thank the entirety of the DISCO team of the LAAS-CNRS. The perpetually good atmosphere was always enjoyable. Thank you Soheib, Audine. Thank you Carine for your help with the water tanks dataset. Thank you Yannick for your warm welcome in the team. Thank you Pauline for your everlasting good mood and valuable insight about any and all topics. I want to thank all PhD students that graduated before me and were my mentors in many aspects, including Adrien, Le Toan, Alexandre, Amaury, Camille, Kévin. I also want to thank all those that will graduate after me for your constant support and friendship, including Charles-Maxime, Lucas, Séna and Rahma. In particular, thank you Charlotte for those three years of sharing the same challenges together. Thank you Rafael, Maxence and Reyan for your support. Finally, thank you Ibis and Léonie for always being there when I needed and for your valuable friendship.

I went studying two months at Linköping University. The whole trip was amazing. I am extremely grateful to my three supervisors Erik Firsk, Mattias Krysander and Daniel Jung for your warm welcome and supervision. I learned so many new interesting notions and acquired useful skills in such a short time. I am very thankful for that. I want to thank the whole Vehicular Systems team, in particular Karin for helping me settle in smoothly and Lars for all the interesting and fun conversations. I still cannot believe how kind and welcoming everyone has been to me. I also want to thank all the PhD students and all the pattern game enjoyers. In particular, Arvind, Oskar, Jian, Ipek, Theodor, Ola, Olov. Thank you Abhi for organizing and inviting

me to the Friday lunches. Thank you Shadi for everything, I wish we play many more fish bowl games in the future. Thank you Arezou and Amina, your office is the best, I wish I had such great neighbors in France. Thank you for all the fun. And finally, thank you Arman for the boardgames and all the great memories we made together.

Finally, I want to thank the people that support me in my everyday life. Thank you to all my friends, your constant support is vital to every aspect of my growth including this thesis. Thank you to my whole family that has always been supporting me, no matter my choices in life. You have had faith in me from the very beginning, and that matters a lot to me. That includes my parents, to whom I am eternally grateful, for everything in my life. My constant happiness and unconcern, I owe to your unconditional support. Ultimately, thank you Fel for being by my side my whole life.

Preamble

This thesis is written in collaboration between Atos and the LAAS-CNRS. I started working at Atos in February 2021 and began the thesis in collaboration with the LAAS-CNRS shortly after. The thesis is founded by an ANRT grant n°2021/0443. My scientific supervisors are Louise Travé-Massuyès and Elodie Chanthery (from the DISCO team at the LAAS-CNRS). My industrial supervisors are Sébastien Delautier and Laurent Garlatti (from Atos Toulouse). This project is related to ANITI within the French "Investing for the Future – PIA3" program under the Grant agreement n°ANR-19-PI3A-0004. During the third year of the thesis, I did a two months mobility to Sweden. I worked with the vehicular systems team, in the department of electrical engineering in Linköping University. In particular, I was supervised by Erik Frisk, Mattias Krysander and Daniel Jung. They are fault diagnosis experts and in particular Erik is the main designer of the Fault Diagnosis Toolbox (FDT) and him and Mattias wrote the code and the algorithms for the toolbox. I used this toolbox extensively in my works and it made everything much easier. They also provided me with the water tanks system that is used as a use case throughout the whole manuscript.

Louis Goupil

List Of Publications

Louis Goupil, Elodie Chanthery, et al. (2022). "A survey on diagnosis methods combining dynamic systems structural analysis and machine learning". In: *33rd International Workshop on Principle of Diagnosis–DX 2022*

Dorian Voydie et al. (2023). "Machine Learning Based Fault Anticipation for 3D Printing". In: *22nd World Congress of the International Federation of Automatic Control (IFAC 2023)*

Louis Goupil, Elodie Chanthery, et al. (2023). "Tree based diagnosis enhanced with meta knowledge". In: *34th International Workshop on Principles of Diagnosis (DX'23)*

Louis Goupil, Louise Travé-Massuyès, et al. (June 2024). "Tree based Diagnosis Enhanced with Meta Knowledge Applied to Dynamic Systems". In: *12th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes*. Ferrara, Italy

Contents

List of Figures

List of Tables

List of Abbreviations

3D	**Three D**imensional
AI	**A**rtificial **I**ntelligence
AMD	**A**dvanced **M**icro **D**evices
ANRT	**A**ssociation **N**ationale de la **R**echerche et de la **T**echnologie
ARR	**A**nalytical **R**edundancy **R**elations
CNN	**C**onvolutional **N**eural **N**etwork
CNRS	**C**entre **N**ational de **R**echerche **S**cientifique
CPU	**C**ore **P**rocessing **U**nit
CUSUM	**CU**mulative **SUM**
DT	**D**ecision **T**ree
DT4X	**D**iagnosis **T**ree **4**(for) e**X**planation
FDI	**F**ault **D**etection (and) **I**solation
FDT	**F**ault **D**iagnosis **T**oolbox
FMSO	**F**ault-driven **M**inimally **S**tructurally **O**verdetermined
GAN	**G**enerative **A**dversarial **N**etwork
GPU	**G**raphics **P**rocessing **U**nit
IFAC	**I**nternational **F**ederation of **A**utomatic **C**ontrol
IMU	**I**nertial **M**easurement **U**nit
KNN	**K** **N**earest **N**eighbors
LAAS	**L**aboratoire d'**A**nalyse et d'**A**rchitecture des **S**ystèmes
LDA	**L**inear **D**iscriminant **A**nalysis
LR	**L**ogistic **R**egression
SA	**S**tructural **A**nalysis
SA-ML	**S**tructural **A**nalysis - **M**achine **L**earning
SO	**S**tructurally **O**verdetermined
ML	**M**achine **L**earning
MLP	**M**ulti **L**ayer **P**erceptron
MSO	**M**inimally **S**tructurally **O**verdetermined
NB	**N**aive **B**ayes
NP	**N**ondeterministic **P**olynomial time
PI-DT4X	**P**hysically **I**nformed **D**iagnosis **T**ree **4**(for) e**X**planation
PLA	**P**oly **L**actic **A**cid
RF	**R**andom **F**orest
sklDT	**S**ci**K**it-**L**earn **D**ecision **T**ree
sklKNN	**S**ci**K**it-**L**earn **K** **N**earest **N**eighbors
sklLR	**S**ci**K**it-**L**earn **L**ogistic **R**egression
sklNB	**S**ci**K**it-**L**earn **N**aive **B**ayes
sklRF	**S**ci**K**it-**L**earn **R**andom **F**orest
sklSVM	**S**ci**K**it-**L**earn **S**upport **V**ector **M**achine
STD	**ST**andard **D**eviation
STL	**S**tandard **T**riangle **L**anguage
SVM	**S**upport **V**ector **M**achine

List of Symbols

DIAGNOSIS	
Σ	system
$n_{\mathbf{z}}$	number of hidden variables
$\mathbf{z}$	set of hidden variables
$n_{\mathbf{x}}$	number of observable variables
$\mathbf{x}$	set of observable variables
$n_{\mathbf{f}}$	number of faults
$\mathbf{f}$	set of system faults
S	set of system states
n_e	number of system equations
e_k	k^{th} system equation
r	residual
G	bipartite graph
A	set of edges of the bipartite graph
ρ	structural redundancy
φ	FMSO set
FS	fault support of an ARR
SM	signature matrix
$\mathcal{R}$	set of model-based ARRs
MACHINE LEARNING	
$\mathcal{D}$	dataset
$\mathcal{D}_{01}$	a dataset made of values in $\{0, 1\}$
$n_{\mathcal{D}}$	number of elements in the dataset
x	sample of the dataset
C	set of classes in a labeled dataset
C_{nom}, C_0	class containing nominal samples
C_{wd}	class of a weakly detectable fault
l	label
T	tree
E	set of edges of the tree
N	set of nodes of the tree
n_0	the root node of the tree
n_i	any node of the tree
$\mathcal{P}$	path of the tree
$\mathbf{A}$	algorithm
SYMBOLIC CLASSIFICATION & REGRESSION	
O	set of operators
c	candidate solution, expression that combines variables from x and operators from O
c_{best}	the retained candidate solution at the end of symbolic classification

w	threshold for symbolic classication
t	classification function
l_{avg}	average length of a candidate solution of symbolic classification
n_g	maximum number of generations for symbolic classification
n_{cs}	number of candidate solutions per generation of symbolic classification
DT4X & PI-DT4X	
d	diagnosis indicator
D	set of diagnosis indicators
$X_r, X_p, X_{T_1}, X_{T_2}$	hyperparameters of DT4X described in Table 4.1
ϵ	hyperparameter of DT4X described in Section 4.2.2.2
p	parsimony coefficient
T1, **T2**, **T3**	conditions required for an expression to be considered a diagnosis indicator
$\mathcal{C}$	time complexity of DT4X
n_n	number of nodes in the tree
$n_{\mathcal{D}}$	number of samples in the dataset
n_C	number of classes in the dataset
n_O	number of input operators
n_x	number of input variables
Ker	the kernel operator
SYSTEMS	
PT	printing time
$n_{windows}$	the number of windows in the 3D printer dataset
T_1, T_2	the water tanks
u_{ref}	input flow of the water tanks system
y_1, y_2, y_3, y_4	signals measured by the water tanks sensors
$d_1, d_2, d_3, d_4, d_5, d_6$	water tanks constants
F_i	fault mode i for the water tanks
f_i	fault signal i for the water tanks
M1, M2, M3, A1, A2	polybox components
MISCELLANEOUS	
$\mathbb{R}$	set of real numbers
$\mathbb{N}$	set of natural numbers
f	function
n, k, K	integers
a	real number
t^*	time
$\vert$	logical OR
&	logical AND
$\neg, \sim$	logical NOT
$\oplus, \wedge$	logical XOR

Chapter 1

Introduction

In contemporary engineering and industrial domains, the ability to promptly and accurately diagnose faults (i.e. detecting and identifying the root cause of a misbehaviour) in complex systems is paramount for ensuring operational efficiency, safety, and cost-effectiveness. The advancement of technology has ushered in an era where traditional diagnostic methods (i.e. relying on the knowledge of an analytical model of the system) are being complemented and, in some cases, supplanted by powerful machine learning techniques (i.e. using data measured on the system). This thesis goal is to combine these two domains, aiming to capitalize on their respective strengths while mitigating their inherent limitations.

1.1 Scientific Goals

Diagnosis methods are often categorized as either model-based or data-driven.

Model-based diagnosis uses a model of the system, most of the time designed from physical laws ruling the behavior of its components, to estimate how that system should behave. The estimated behavior is then compared with the actual behavior of the system. Such a model is not always available, in particular for complex systems or systems protected by business secrets.

Data-driven diagnosis methods, often based on machine learning, stand on algorithms able to diagnose the system without formalized knowledge about it. These algorithms are used on data recorded from the system — usually with sensors. However, data-driven methods require large amounts of data and often lack explainability as to why a fault occured.

The scientific goal of this thesis is to design one or more diagnosis methods that mix model-based and data-based approaches. Data is often easy to obtain (even more nominal data). Model-based algorithms give explainability, but they require a precise physical, analytical model of the system. They often allow to take advantage of expert knowledge of the system in order to improve the accuracy of diagnoses. The goal is to design a method that relies on data, but that gives explainability in order to correct faults. Intrinsically mixing model-based and data-based methods should allow to obtain explainability and enhance data-based diagnosis with expert knowledge. However, it is important to find a method that does not combine the limitations of each approach, but rather palliates them.

Overall, this means that we target a data-driven diagnosis method that:

- learns system information from the data;
- uses meta-knowledge (i.e. knowledge about the way the method works) from model-based methods;
- allows for expert knowledge to be exploited to enhance diagnosis;

- is explicable (i.e. allows to identify what causes a fault to occur in order to be able to correct it).

1.2 Industrial Context

Industrial applications are driven by the imperative to enhance operational efficiency, reliability, and safety of operational systems. In many fields such as aerospace, automotive, and others, the quality of the products manufactured and utilized must meet very high standards. Faults have to be extremely scarce in order to meet very strict safety requirements.
Many standards govern this quality, such as the AS/EN9100 series standards, which draw inspiration from the ISO 9001 Quality Management standard. Thus, to comply with the numerous standards in these domains, manufacturers must, among other things, implement a system for regularly monitoring and controlling the quality of their parts and systems. The repercussions of non-conformities can be numerous and more or less severe, ranging from a decrease in the brand image for the company responsible for the defect, to technical issues that can lead to various accidents.

Nowadays, systems become more and more complex. For instance, they can be large, heterogeneous, uncertain, hybrid, dynamic and non-linear. Examples surround us everyday: 3D printers, autonomous cars, aircraft subsystems, smartphones, computers and all their subsystems, etc... This complexity, combined with business secrets, leads to the study of not fully comprehended systems. For that reason, there is a need for diagnosis tools that fit as many system types as possible and in particular large non linear dynamic systems for which no full physical model is available, only some partial expert knowledge. The integration of machine learning and model-based diagnosis holds great promise for industrial applications because of the ability to take advantage of expert knowledge accumulated along the operational history of the system but also of the data that can be measured on the system and exploited with machine learning.

A recent transformative shift in manufacturing paradigms can be observed with the widespread use of 3D printing. This rise is propelled by technological advancements, offering unparalleled design freedom and rapid prototyping capabilities. 3D printing ability to customize and personalize products, coupled with supply chain resilience and sustainability benefits, has made it a transformative force across various sectors. However, those tremendous advancements hide some harsh reality: 3D printing is by no means reliable. Defects[1] systematically appear on parts, no matter the technology, be it additive manufacturing, particle fusion or stereolithography. No technique allows for a hundred percent success rate during printing. For small-scale 3D printers using plastic filament, the impact might be minimal, wasting only a small amount of time and plastic. However, for metal fusion printers, a single day of print can cost millions of euros. Hence the need for a diagnosis method able to correctly detect if a defect occurs during the printing process, or better, predict whether a defect will appear during printing in the future.

This thesis focuses on diagnosis of technological systems and it is largely interested in experiments performed on a 3D printer as proposed by our industrial partner. For the aforementioned reasons, the eventual goal, in the context of 3D printing, is prognosis. Exploring early diagnosis possibilities is a step in that direction.

[1] The term 'defect' is used to indicate a problem with the printed part, while 'fault' is reserved for problems related to the systems or processes.

1.3 Manuscript Organization

Contrary to what is done traditionally, in this manuscript, we took the decision to spread out the state of the art in every chapter. This makes for more self-sufficient chapters, that start with background concept and notations along with the associated state of the art, followed by the description of methods using these concepts, again followed by the application of these methods on use cases. Then, each chapter contains conclusions and perspectives about the presented works.

Chapter 2 on page 7 goal is to solely use data-driven techniques for diagnosis in order to study the drawbacks and advantages of such methods, and also to have a reference to compare with for future studies. It starts by introducing machine learning and a few algorithms and then presents a state of the art of 3D printing diagnosis methods. Finally, it introduces the 3D printer use case and displays the results of machine learning algorithms applied to this use case.

Chapter 3 on page 29 introduces the concept of structural analysis and gives a state of the art of methods combining machine learning with model-based diagnosis. Next, it illustrates such a method on the water tanks use case. The aim of this chapter is to look at what has already been done in the domain of combined data-driven and model-based diagnosis and to propose an enhanced version of what already exists. Then, by reflecting on this experiment, the goal is to develop a novel data-driven diagnosis method that relies on meta-knowledge from model-based.

Chapter 4 on page 45 presents DT4X, the model-based inspired data-driven diagnosis algorithm we developed as an answer to both the industrial and scientific goals. The background required for DT4X is first introduced, then the principle of the algorithm is explained and some properties of the algorithm are studied.

Chapter 5 on page 69 illustrates the use of DT4X to diagnose four different types of systems, including the water tanks and the 3D printer. A special emphasis is put on the application of DT4X to logic circuits because it allows to find interesting logical relations, which, as far as we know, have never been identified before.

Chapter 6 on page 91 presents PI-DT4X, an alternative to DT4X that uses the structural model of the studied system in order to orient the training. The goal is to explore whether it is possible to drastically improve the quality of the explanation gathered from DT4X by feeding more knowledge about the studied system. PI-DT4X is then tested on two systems.

Finally, Chapter 7 on page 109 concludes the manuscript by giving a short summary of the research, recapping what has been answered w.r.t the initial problem and providing some global perspectives for future works.

1.4 Useful Concepts and Notations

In most sections, methods are presented in a generic scenario, without any specific system in mind. Here, notations and notions are defined for the whole manuscript, for coherence sake. Please, refer to this section when doubting the meaning of a symbol (along with the list of symbols at the beginning of the manuscript).

1.4.1 The System

Definition 1 (System)**.** *A system* Σ *is a set of interconnected or interdependent components working together to achieve a common goal or function.*

Definition 2 (Non-observable Variables)**.** *Non-observable variables, or hidden variables, or internal states, are physical quantities characterizing the system that are not measured by the instrumentation in place on the system.*

Definition 3 (Observable Variables)**.** *Observables variables, or measurable variables, are physical quantities characterizing the system that are measured by the instrumentation in place on the system.*

We consider a system Σ. This system is composed of $n_{\mathbf{z}}$ non-observable variables and $n_{\mathbf{x}}$ observable variables. The vector of non-observable variables is denoted $\mathbf{z} = (z_1, \ldots, z_{n_{\mathbf{z}}}) \in \mathbb{R}^{n_{\mathbf{z}}}$ and the vector of observable variables is denoted $\mathbf{x} = (x_1, \ldots, x_{n_{\mathbf{x}}}) \in \mathbb{R}^{n_{\mathbf{x}}}$[2].

Definition 4 (Fault)**.** *In the context of diagnosis, a fault refers to an abnormality, defect, or malfunction within the system that leads to its improper or sub-optimal functioning. Faults can manifest in various ways, such as errors in operation, deviations from expected behavior, or failures to meet performance requirements.*

Definition 5 (State of a System)**.** *The state of a system refers to the condition or configuration of a system at a particular moment in time. It represents the set of values or parameters that fully describe the system behavior and characteristics at that specific instant. A state can be nominal if the system is behaving as it is supposed to according to its purpose. Otherwise, the state of the system is said to be faulty. A system cannot be in two different states at the same time.*

Σ is subject to certain faults. We consider $n_{\mathbf{f}} \in \mathbb{N}$ different faults. Thus, Σ can either be in a nominal state or in one of the faulty states. The set of states is denoted $S = \{nominal, f_1, \ldots, f_{n_{\mathbf{f}}}\}$[3]. The vector of faults is denoted $\mathbf{f} = (f_1, \ldots, f_{n_{\mathbf{f}}})$.

Definition 6 (Ambiguous State of a System)**.** *A system is said to be in an ambiguous state when its state is not exactly known. An ambiguous state is a logical statement expressing a disjunction of states.*

For instance, if Σ is diagnosed as being either subject to f_1 or f_2 it is said to be in an ambiguous state. On the contrary, if Σ is said to be subject to f_1, its state is not ambiguous.

The system model $\Sigma(\mathbf{z}, \mathbf{x}, \mathbf{f})$ is a set of differential or algebraic equations $e_k(\mathbf{z}, \mathbf{x}, \mathbf{f})$, $k \in [1, n_e]$ with n_e the number of equations.

1.4.2 Diagnosis

In this manuscript, *diagnosis* always refers to fault diagnosis. It is not to be mixed-up with medical diagnosis, even though the concepts are very similar.

Definition 7 (Fault Diagnosis)**.** *Fault diagnosis is the process of detecting the presence of faults within a system or process, determining their root cause or causes, and providing information about the nature and severity of the faults.*

Diagnosis often consists in different steps: detection, isolation and identification of the fault.

[2] This goes against the traditional diagnosis notations that consist in having $\mathbf{z}$ represent the observable variables and $\mathbf{x}$ the hidden variables. This choice has been made in order to coincide with traditional machine learning notations that use x as the sample.

[3] f_i designates a faulty state. It can be multiple faults happening at the same time, it does not have to only be one fault.

Definition 8 (Detection)**.** *Detection is the process of identifying the presence of abnormalities, deviations, or malfunctions within a system or process. It involves detecting when the system's behavior deviates from its expected or normal operating conditions. The primary goal of fault detection is to recognize the existence of faults as soon as possible, allowing for timely intervention to prevent or mitigate potential adverse effects.*

Definition 9 (Detectability)**.** *A fault is said to be detectable when the set of sensors in place on the system allow for its detection.*

Definition 10 (Isolation)**.** *Isolation is the process of identifying and pinpointing the specific component, subsystem, or area within a system where a fault or malfunction has occurred. It involves narrowing down the possible sources of the fault to isolate the root cause accurately.*

Definition 11 (Isolability)**.** *A specific fault (i.e. a certain component malfunction) is said to be isolable when the set of sensors in place on the system allow to identify that this specific fault is present when a detection of it occurs. This means that when this fault occur, it can not be confused with another fault. When two faults are said to be isolable, it means that these two faults make the system behave in different ways, and thus can not be confused with one another. It is also often referred to as diagnosability.*

Definition 12 (Full Isolability)**.** *A fully isolable system is a system where all faults are isolable (w.r.t. the set of sensors). It is also often referred to as full diagnosability.*

Definition 13 (Identification)**.** *Identification is the process of identifying the specific cause or source of the fault.*

Definition 14 (Diagnosability of a System)**.** *The diagnosability of a system is whether the possible faults that can occur in this system are diagnosable or not.*

Definition 15 (Maximum Diagnosability of a System)**.** *The maximum diagnosability of a system is reached when all isolable faults are isolated.*

Chapter 2

Machine Learning Based Diagnosis

In order to design a diagnosis method using a data-driven approach, it is important to understand the processes of such an approach. In particular, machine learning is a widespread source of data-driven approaches. This chapter aims at giving some background on machine learning by, notably, giving examples of the most classic machine learning algorithms and their principle. Then, it provides a state of the art of 3D printing diagnosis methods and, lastly, it applies machine learning to a specific 3D printer whose diagnosis is part of the industrial goal of this thesis. The objective is firstly to highlight the drawbacks of using machine learning exclusively, and then to study how it can be improved by using model-based approaches in conjunction with it. A second objective is to obtain results that can serve as benchmarks for comparison with the other methods developed in this thesis.

2.1 Background

First, we define the concepts and notations that surround a dataset. A dataset is a key part of machine learning. Then, the core principles of machine learning are described and a few classic algorithms are presented. These are used throughout the manuscript.

2.1.1 Dataset

Definition 16 (Data-based Algorithm)**.** *A data-based (or data-driven) algorithm is a type of algorithm that uses data as its primary input or relies heavily on data-driven techniques to make decisions, solve problems, or perform tasks. These algorithms typically leverage large datasets to learn patterns, extract insights, or make predictions.*

Definition 17 (Dataset)**.** *A dataset $\mathcal{D}$ is a structured collection of data that is organized and stored in a specific format for analysis, processing, or presentation. It typically consists of individual data points or records, each containing one or more attributes or variables.*

Definition 18 (Sample)**.** *In the context of data-driven algorithms, a sample x refers to a single instance or observation within the dataset $\mathcal{D}$. It is also called a data point or an observation.*

In this manuscript, many data-based algorithms are described and used. In order to train those algorithms, a dataset $\mathcal{D}$ is required. The size of $\mathcal{D}$ is $n_{\mathcal{D}}$. If $\mathcal{D}$ is a labeled dataset, it is made of $n_{\mathcal{D}}$ pairs. A pair is made of a sample x and a label l. x can take many forms, such as an image, a text, a matrix, a vector, a real number, etc.

In our case, unless specified, it represents the values of the system observable variables $\mathbf{x} = (x_1, \ldots, x_{n_\mathbf{x}})$ as defined in Section 1.4.1 on page 3 at some time point t^*. A sample is hence a vector of values $x = \mathbf{x}(t^*) = (x_1(t^*), \ldots, x_{n_x}(t^*)) \in \mathbb{R}^{n_x}, n_x \in \mathbb{N}$. The set of possible classes is denoted C. The label $l \in C$. Also, in this manuscript, unless otherwise specified, the set of classes is the set of possible system states: $C = S$. In the case where $\mathcal{D}$ is not labeled, it simply contains the $n_\mathcal{D}$ samples, without corresponding labels.

In this manuscript, the term *sample* corresponds to a data point, an observation of the system at a given time (described by its variables but also sometimes the derivatives which give an indication of the evolution of the system over time).

2.1.2 Machine Learning

Machine learning (Alpaydin, 2021) is a subfield of artificial intelligence that focuses on the development of algorithms and computational models capable of learning from data and making predictions or decisions based on that learning.

In this manuscript, we only consider supervised learning. This means that data is labeled, each sample x has a corresponding label l that is used during the training step. We limit our scope to systems for which a labeled dataset can be built from the system in operation.

2.1.2.1 General Principles

Machine learning involves the following key steps, in order:

1. **Data Collection**. The first step in machine learning is to gather relevant data that contains information about the problem to solve. This data can come from various sources such as sensors or databases.

2. **Data Preprocessing** (García et al., 2016). Once the data is collected, it often needs to be cleaned and preprocessed to remove noise, handle missing values, and normalize or scale the features. This step ensures that the data is in a suitable format for training machine learning models.

3. **Feature Engineering** (Zheng and Casari, 2018). Feature engineering involves selecting, transforming, or creating new features from the raw data to improve the performance of machine learning models. This may include extracting useful information from the data, combining features, or reducing dimensionality.

4. **Algorithm Selection**. An appropriate machine learning algorithm that is well-suited to the problem at hand has to be selected. The choice depends on factors such as the type of problem (e.g., classification, regression, clustering), the nature of the data, and the desired performance metrics.

5. **Training**. In supervised learning, the selected algorithm is trained on a (often random) subset of the labeled dataset, where each data point is associated with a target label or outcome. During training, a model is learned from the input-output pairs in the training data and the algorithm adjusts its parameters to minimize a loss function, which measures the difference between the predicted and actual outputs. Often, during traing, a K-fold cross validation (Refaeilzadeh, Tang, and H. Liu, 2009) process is used ($\mathrm{K} \in \mathbb{N}$). It consists in separating the training set in K subsets that are used interspersed as validation sets in order to check that what is learned by the algorithm can be generalized. The principle is illustrated in Figure 2.1 on the facing page.

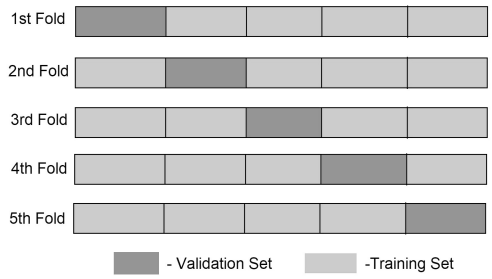


FIGURE 2.1: Example of a 5-Folds Cross Validation

6. **Testing**. After training, the performance of the model is evaluated on a separate testing dataset to assess how well it generalizes to unseen data. This evaluation helps to identify any issues such as overfitting (where the model performs well on the training data but poorly on new data) or underfitting (where the model fails to capture the underlying patterns in the data).

7. **Hyperparameter Tuning** (L. Yang and Shami, 2020). Machine learning algorithms often have hyperparameters that control their behavior, such as the learning rate, regularization strength, or the number of layers in a neural network. Hyperparameter tuning involves selecting the optimal values for these hyperparameters to improve the performance of the model.

Some algorithms require specific additional steps. Feature engineering is not mandatory and it depends on the use case.

Definition 19 (Training Dataset)**.** *A training dataset is a portion of data that is used to train a machine learning model during the training step. It consists of input samples x along with their corresponding target labels l. It usually represents between* 70% *and* 90% *of the dataset.*

Definition 20 (Testing Dataset)**.** *A testing dataset is a portion of data that is used to test a machine learning model during the testing step. It consists of input samples x along with their corresponding target labels l. It usually represents between* 10% *and* 30% *of the dataset.*

Separating the dataset into a training and a testing set is usually done during the preprocessing step.

The purpose of machine learning varies depending on the use case. The most common applications are classification (Kotsiantis, Zaharakis, Pintelas, et al., 2007) and regression (Rong and Bao-Wen, 2018) tasks.

Definition 21 (Classification)**.** *Classification refers to the process of predicting a categorical label or class l for a given input data point x. The goal is to assign each sample x to one of a predefined set of classes or categories based on its features or attributes. A classification algorithm can be seen as a function f that maps the sample space (often $\mathbb{R}^n, n \in \mathbb{N}$) to the class space ($C$). $f(x) = l$.*

Definition 22 (Regression)**.** *Regression refers to the process of predicting a numerical value $a \in \mathbb{R}$ for a given input data point x using a function f. Unlike classification, which predicts categorical labels or classes, regression models aim to estimate a quantity that can vary over a continuous range. A regression algorithm can be seen as a function f that maps the sample space (often $\mathbb{R}^n, n \in \mathbb{N}$) to a continuous space ($\mathbb{R}$). $f(x) = a$.*

2.1.2.2 Classic machine learning algorithms

Some machine learning algorithms are mentioned throughout the manuscript. Most are just used as a comparison to evaluate performance of the proposed algorithms. They mainly consist in classification rather than regression. Others, such as decision trees symbolic classification and symbolic regression, are detailed in depth in further sections where they are used. Here is a quick overview of the different algorithms that gives a basic understanding of how they work and what is their purpose. Some additional information is given in Table 2.1 on page 12. Singh, Thakur, and Aakanksha Sharma, 2016 gives a detailed overview of many machine learning algorithms including those presented here.

Logistic Regression (LR) Logistic regression (Kleinbaum et al., 2002) is a statistical method used for binary classification tasks, where the goal is to predict the probability that a given input data point belongs to one of two classes or categories. Despite its name, logistic regression is a classification algorithm rather than a regression algorithm, as it predicts discrete class labels rather than continuous values. Logistic regression models assume a linear decision boundary separating the classes in the feature space. This decision boundary divides the feature space into regions where one class is more likely than the other. Being a linear model, it is limited in its performances because of its low number of parameters compared to other approaches.

Linear Discriminant Analysis (LDA) This statistical method (Izenman, 2013) finds a linear combination of features that characterizes or separates two or more classes in windows. It is the supervised equivalent of a principal component analysis (Greenacre et al., 2022). It projects the dataset along linear discriminant axes and represents visually one-vs-all axes for all classes. It gives a quick, visual and efficient way to check if the data allows to separate classes or not.

Decision Trees (DT) A decision tree (Kotsiantis, 2013) is a predictive modeling technique used in machine learning and data mining for both classification and regression tasks. It represents a flowchart-like structure where each internal node represents a decision based on the value of a feature attribute, each branch represents the outcome of the decision, and each leaf node represents the final decision or prediction. Decision trees are precisely presented in Section 4.1.3 on page 48.

Random Forest (RF) A random forest (Rigatti, 2017) is an ensemble learning technique (i.e. it merges predictions from multiple models) for both classification and regression tasks. It is composed of multiple individual decision trees, each trained on a random subset of the training data and using a random subset of the input features. The process of combining multiple decision trees in a random forest reduces variance and improves predictive performance compared to a single decision tree. This is because the individual trees may have different biases and errors, but when combined, they tend to cancel out each other's shortcomings. The predictions of individual decision trees in the random forest are combined to make the final prediction. For classification tasks, the mode (most frequent class) of the predictions is typically used, while for regression tasks, the average of the predictions is taken.

K-Nearest Neighbors (KNN) K-Nearest Neighbors (Peterson, 2009) is an algorithm used for both classification and regression tasks. It is a non-parametric (i.e. the model structure is not given *a priori* but learned from data) and instance-based learning method, meaning it makes predictions based on the similarity of input data points to known examples in the training dataset. KNN makes predictions by identifying the K ($K \in \mathbb{N}$) nearest neighbors of a given query data point in the feature space. The class label (in classification) or the numerical value (in regression)

of the query point is then determined by the majority class or the average value, respectively, of its nearest neighbors.

Multi-Layer Perceptron (MLP) A multilayer perceptron is a type of feedforward artificial neural network (Yegnanarayana, 2009) consisting of multiple layers of nodes (neurons), each layer fully connected to the next layer. MLPs are widely used for supervised learning tasks such as classification, regression, and pattern recognition (Riedmiller and Lernen, 2014). The input layer of an MLP consists of neurons representing the input features of the dataset. Each neuron corresponds to one feature, and the values of these neurons are passed to the neurons in the next layer. Between the input and output layers, MLPs can have one or more hidden layers. Each hidden layer consists of multiple neurons that perform computations on the input data using weighted connections and activation functions. The output layer of an MLP produces the final predictions or outputs of the model. The number of neurons in the output layer depends on the task at hand; for example, in binary classification tasks, there may be one neuron representing the probability of belonging to one class, while in multi-class classification tasks, there may be multiple neurons representing the probabilities of belonging to each class. The connections between neurons in adjacent layers are characterized by weights, which represent the strength of the connection. Each neuron also has an associated bias term, which allows the model to learn nonlinear relationships in the data. During training, the weights and biases of the MLP are adjusted through the process of backpropagation to minimize a loss function, such as mean squared error or cross-entropy loss (Raul Rojas and Raúl Rojas, 1996).

Support Vector Machine (SVM) A support vector machine (Steinwart and Christmann, 2008) is a supervised algorithm used for classification and regression tasks. SVM aims to find the hyperplane that best separates the data points of different classes while maximizing the margin, which is the distance between the hyperplane and the closest data points (support vectors). By maximizing the margin, SVM seeks to achieve better generalization performance and robustness to noisy data.

Naive Bayes (NB) Naive Bayes (Webb, Keogh, and Miikkulainen, 2010) is a probabilistic classification algorithm based on Bayes' theorem (Joyce, 2003), with a naive assumption of feature independence. This assumption is that the features are conditionally independent given the class label. This means that the presence or absence of a particular feature is assumed to be unrelated to the presence or absence of any other feature, given the class label. To make predictions, Naive Bayes calculates the posterior probability of each class given the observed input features using Bayes' theorem. The class with the highest posterior probability is then predicted as the output class label.

2.2 State of the Art of 3D Printing Diagnosis Methods

The industrial goal of this thesis is, in particular, to develop a diagnosis method to improve the quality of 3D printed parts. In the literature, many aspects of this problem are explored, from performing predictive maintenance on the printer to looking at each layer of the part and ensuring its good quality. We have identified a few recent articles that tackle the problem with various angles and solutions in order to get a somewhat comprehensive look at the existing methods. As the field is ever growing, we have added new papers that got published while the thesis was on-going. They are recapped in Table 2.2 on the next page.

	Explicable	Scalable	Further Documentation
LR	No	Yes	LR
LDA	Partly	Yes	LDA
DT	Partly	Yes	DT
RF	No	Yes	RF
KNN	Partly	Yes	KNN
MLP	No	Yes	MLP
SVM	No	Hardly	SVM
NB	No	Yes	NB

TABLE 2.1: Additional Information about the Classic Machine Learning Algorithms. The further documentation includes detailed explanations about the influence of hyper-parameters.

Paper	Data-based	Real-time	Correction	Image-based	Anticipation
Baumann and Roller, 2016		×		×	
Brion and Pattinson, 2022	×	×	control loop	×	
Delli and Chang, 2018	×		Stops when fault appears	×	
He et al., 2018	×				
Z. Jin, Z. Zhang, and Gu, 2019	×		Only under/over extrusion	×	
Loja et al., 2020	×				
Uhlmann et al., 2018	×				
Yen and Chuang, 2022	×	×			
S. Zhang et al., 2021	×				

TABLE 2.2: Details on Some Fault Diagnosis Methods for 3D Printing

Paper	Algorithm	Preprocessing	Accuracy
Brion and Pattinson, 2022	Multi-head neural network	Data augmentation, image resizing	96.6%
Delli and Chang, 2018	SVM kernel trick	Divide, resize and recolor images	
He et al., 2018	least square loss SVM		94.44%
Z. Jin, Z. Zhang, and Gu, 2019	CNN (Resnet)	Concatenation of images	
Loja et al., 2020	GAN	Non linear mapping to preclassification space	
Uhlmann et al., 2018	Clustering with k-means	Normalization	
Yen and Chuang, 2022	Neural Network with only 1 hidden layer		83.5%
S. Zhang et al., 2021	Reservoir computing	feature extraction using aggregation function	98%

TABLE 2.3: Insight Into Algorithms Used for Fault Diagnosis of 3D Printing. Empty cells correspond to unavailable information.

Most methods rely on data. The only paper (Baumann and Roller, 2016) not using a data-driven technique is using image processing. That is not even a model-based method either and it can only be used to detect faults, not correct them. None of the methods try to anticipate faults. Some try to exploit images, with Baumann and Roller, 2016 making use of image processing to detect a fault in real time. The others analyze the video or the images post-printing and try to determine whether a fault occurred. Correction is considered when real-time detection is possible or when detection is made by pausing the print (Delli and Chang, 2018, Z. Jin, Z. Zhang, and Gu, 2019). The only work that applies a full real-time correction is Brion and Pattinson, 2022. It uses detection values to compute a correction to apply to the main printing parameters using a multi-head neural network that uses images automatically labelled by deviation from optimal printing parameters. The correction is then sent through gcode to the printer.

As most use data-driven techniques, it is insightful to look at which algorithms are used. A quick description of the algorithms is given in Table 2.3. The given accuracy has to be taken with a grain of salt because it heavily depends on which fault types the papers try to identify, the quality of data and the evaluation metric.

For further information on machine learning applied to 3D printing diagnosis, we identified three interesting surveys of this domain: Wang et al., 2020, Ademujimi, Brundage, and Prabhu, 2017 and Goh, Sing, and Yeong, 2021. They mainly focus on fault detection from various machine learning and deep learning methods. Goh, Sing, and Yeong, 2021 goes as far as to study the impact of designing the part, tuning the material and even wonders about cybersecurity of a 3D printing process. One of the main takeaways is that machine learning rarely gives an explanation of why a fault is occurring and that it hinders the correction possibilities.

To summarize, almost no algorithm is able to reliably and automatically predict or detect faults in real-time. The only one that performs real-time detection and

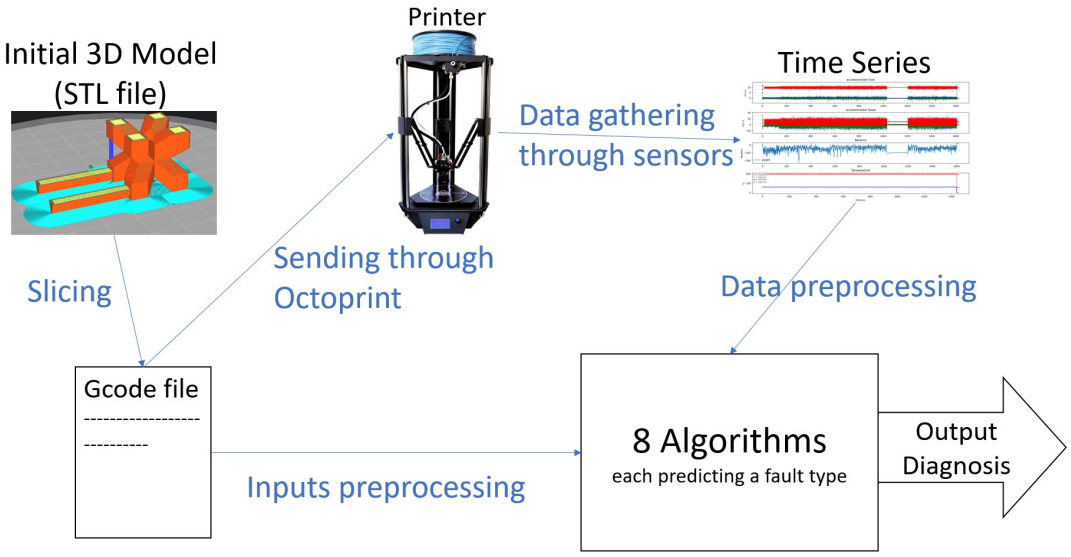


FIGURE 2.2: Summary of the Study

correction with a very high accuracy is Brion and Pattinson, 2022. It solely relies on data-based methods, allows for some expert knowledge to be used to enhanced diagnosis and only gives a partial explanation as to why a fault occurs or learn system information (which are the criteria we set in Section 1.1 on page 1). In this manuscript, our goal is to take advantage of model-based methods to incorporate some knowledge about the system and outperform data-driven only methods. But first, let us study how traditional machine learning methods perform on our 3D printer use case.

2.3 Machine Learning Applied to the 3D Printer

The goal of this section is to study how data-driven approaches perform on a complex dynamical system such as a 3D printer. In particular, it is interesting to study the drawbacks of such methods in order to then develop a novel approach that uses model-based theory in order to palliate those drawbacks. It can also serve as a benchmark for then comparing with a hybrid diagnosis method.

Specifically, on the considered 3D printer (see Section 2.3.1), the goal is to diagnose defects **on the printed parts**, not faults in the printer. A fault designates a defect on the printed part, in the context of this system. Figure 2.2 recapitulates how printing works, data is acquired and a diagnosis is established. The following sections detail each step of this figure, including the printing process and how a diagnosis is established.

2.3.1 System Description

The studied system is a physical 3D printer present at Atos. As such, we can use it to experiment, modify the sensing equipment and print a dataset whenever needed. It is a microdelta rework printer developed by eMotion Tech. A picture of the printer is given in Figure 2.3 on the facing page. This printer uses plastic additive manufacturing, meaning it successively deposes thin layers of melted *Poly Lactic Acid* (PLA) at the right coordinates to build the requested part. PLA is a kind of bioplastic

FIGURE 2.3: 3D Printer

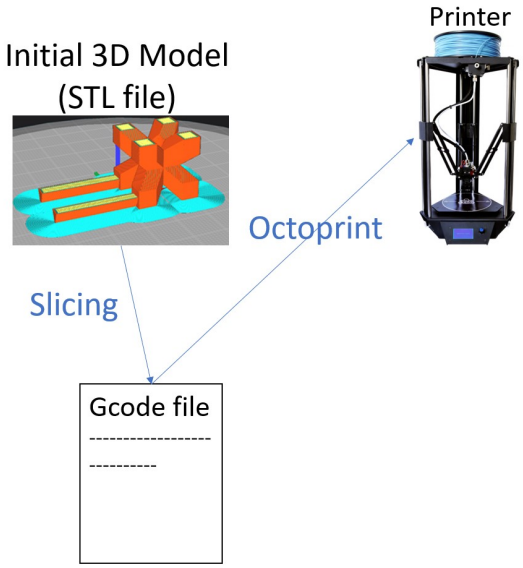
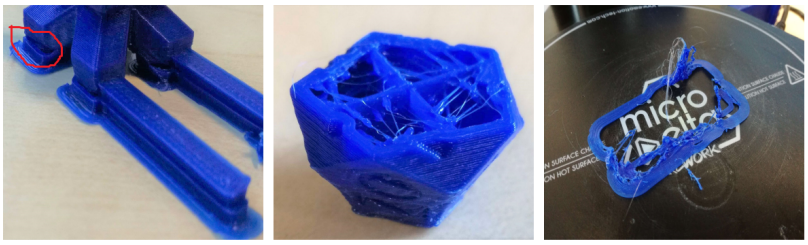


FIGURE 2.4: Print Process

(plastic made from renewable, organic sources) that is used in the most common type of printers, see Garlotta, 2001. Its melting temperature is between 170°C and 180°C.

As illustrated in Figure 2.4, in order to print a part, a 3D model of this part has to be designed. Then, this model is converted into a *Standard Triangle Language* file (STL file for short, originally STereoLithography, see Bártolo, 2011). This file describes only the surface geometry without any representation of color, texture, scale or units. This file is then fed to a *slicer*, a software parameterized with specific printer characteristics, that outputs a file understandable by the 3D printer. In our case, this file is a *gcode* file written in the language Marlin (Krüger et al., 2018). This file is then given to the printer that prints the part. This is all summarized in Figure 2.4. The web interface Octoprint is used to send the gcode to the printer through an ssh (secure shell protocol) connection.

The process of slicing is crucial. It is at this stage that most printing parameters are set, and, based on 3D printing experts' experience, it is at this stage that most



Layer Shifting Not Extruding No Adhesion

FIGURE 2.5: Critical Faults

common printing faults are generated. For instance, the printer nozzle temperature and printing speed are set during slicing. Requiring a nozzle temperature of 220°C with a speed of 10 mm/s would result in a failed print with overheating of the part or blob appearances since the heat would melt the PLA faster but the nozzle would move too slowly in comparison. One of the goals of this study is to be able to detect such bad parameterization and correct it before it leads to a fault by altering in real-time the gcode sent to the printer (the Octoprint interface allows to send gcode lines in real time).

The physical model of the 3D printer is not fully known, meaning that we cannot use system equations to model the printer behavior. Hence the use of methods based on data that do not require system knowledge, such as machine learning. This also means that we do not know precisely how the printing parameters influence the behavior of the system. Therefore, when deciding those parameter values for building the dataset, we have to be careful not to be biased by what we intuitively think are the correct parameters.

2.3.2 3D Printer Fault Types

3D printing having poor performances overall, many people got interested in identifying and classifying possible faults. For this study, it is important to identify the faults we want to be able to diagnose.

Based on the feedback of experts, eight different types of faults have been selected, divided in 2 categories:

- Critical faults (see Figure 2.5);
- Severe faults (see Figure 2.6 on the facing page).

When a critical fault occurs, the printed part is expected to be non-functional and so distorted that it becomes useless to continue the print. However, when a severe fault occurs, the part remains functional but its aspect integrity is questioned. Obviously, depending on the aim of the part and the requirements of the part manufacturer, these categories may vary.

Here is a short description of the possible faults:

- Layer Shifting: a shift in the printing coordinates induces a shift between two layers of the printed part. This is often caused by the engine not correctly driving the belt and skipping a step on one of the three belts of the printer, resulting in the printing coordinates being slightly shifted.
- Not Extruding: the nozzle stops delivering PLA. This mostly occurs when the spool is empty or when the wire is stuck.

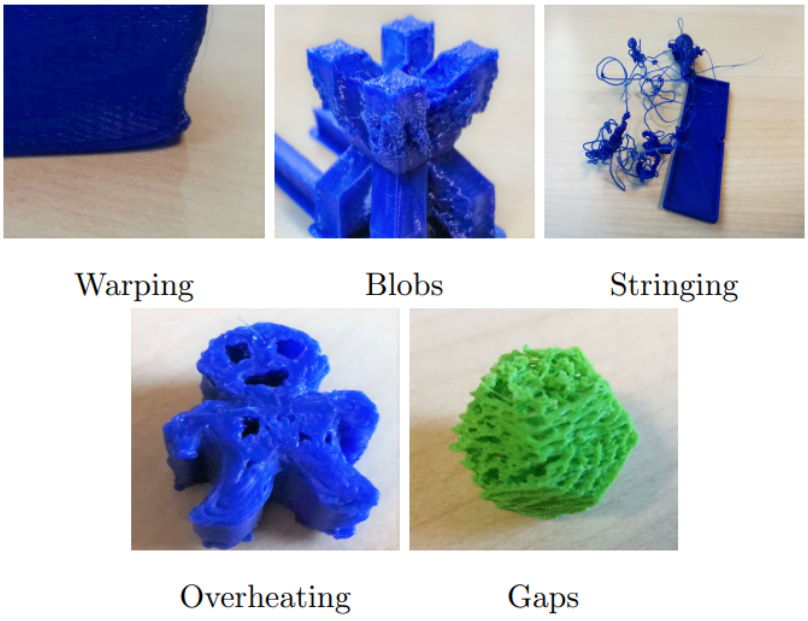


FIGURE 2.6: Severe Faults

- No Adhesion: the PLA does not stick to the board. This mostly occurs when the board temperature is too low or too high with respect to the nozzle temperature.
- Warping: an angle of the part in contact with the board gets distorted until it is not anymore in contact. This mostly happens when the board temperature is too high.
- Blobs: small blobs appear on the surface of the part. This often occurs when printing speed is too low, too much PLA gets deposited and creates a surplus.
- Stringing: thin filaments spread from the part. This often occurs at the end of the print, when the nozzle goes back to its original position. The filaments are made of leftover PLA that leaks from the nozzle.
- Overheating: the part appears melted. This often occurs when the nozzle temperature is too high, the PLA is almost liquid and does not stay where it should.
- Gaps: gaps appear on the surface of the part. This mainly happens when the printing speed is too high, the PLA does not have time to escape from the nozzle.

2.3.3 Measuring Equipment

The goal of this study is to know whether it is possible to predict faults in real-time during a print with enough accuracy so that the removal of the detected faults would drastically improve the average quality of 3D printed parts. A specific interest is shown in the ability to anticipate faults using data from sensors (that record during printing) and knowledge about the part to be printed in the form of the gcode file.

In order to gather this data, the system is equipped with multiple sensors. Since we suspect the three factors affecting printing to be initial, environmental and printing parameters, the inputs of the machine learning models are as follows:

- 3D model of the part to be printed in the shape of gcode;

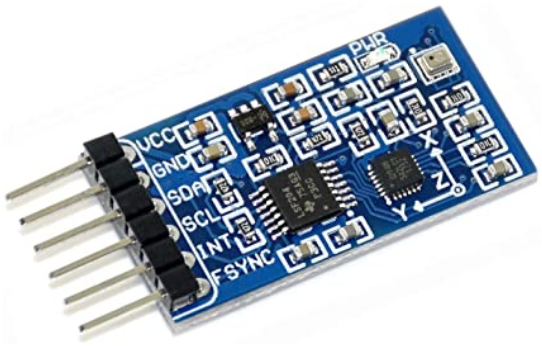


FIGURE 2.7: Inertial Measurement Unit

- sensor values of printer parameters recorded during printing;
- sensor values of environmental parameters recorded during printing.

We equipped the printer with sensors according to the above items. The printer also has integrated sensors. All the signals retrieved (see Table 2.4 on page 20 for a summary) are:

- From the printer integrated sensors:
  - Nozzle Temperature (actual and target);
  - Bed temperature (actual and target);
  - Layer information (number, time, mesh, etc.);
  - General printing settings of maximum speeds, maximum accelerations, dimension boundaries, printing time, nozzle travel lengths, etc.;
- From an inertial measurement unit (IMU) placed close to the nozzle (Figure 2.7):
  - Nozzle acceleration, angular speed and exterior temperature;
- From an accelerometer placed close to the board (Figure 2.8 on the next page):
  - Board acceleration;
- From a weight placed below the wire spool (Figure 2.9 on the facing page):
  - Wire spool weight, equivalent to the wire tension in our case.

The pin mapping is given in Appendix A on page 113. Some other instrumentation was set up, but not used in the dataset. This additional instrumentation is also described in Appendix A on page 113.

All the sensors are linked together through a local network. It makes the trigger of events to synchronize the signals easier and allows us to monitor the process using a laptop connected to the network. The whole setup is presented in Figure 2.10 on page 20. All these sensors allow to collect data.

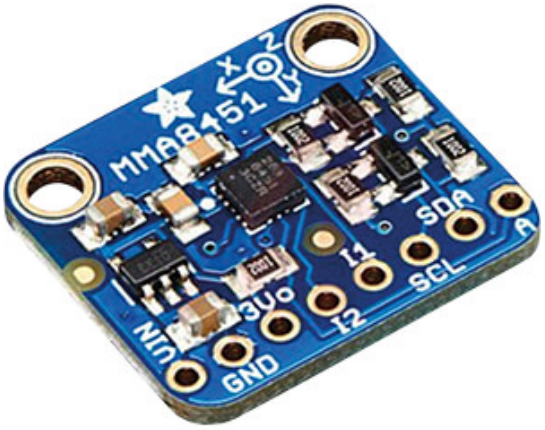


FIGURE 2.8: Board Accelerometer

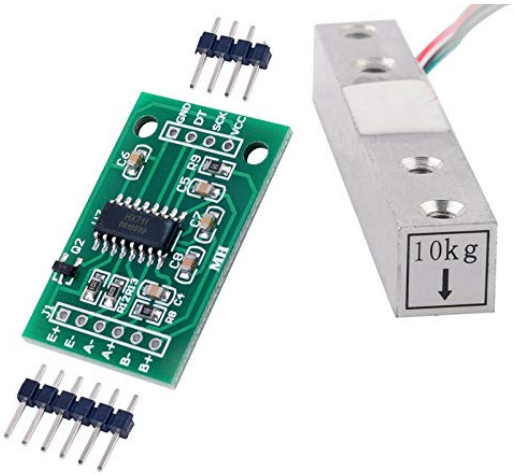


FIGURE 2.9: Wire Spool Weight

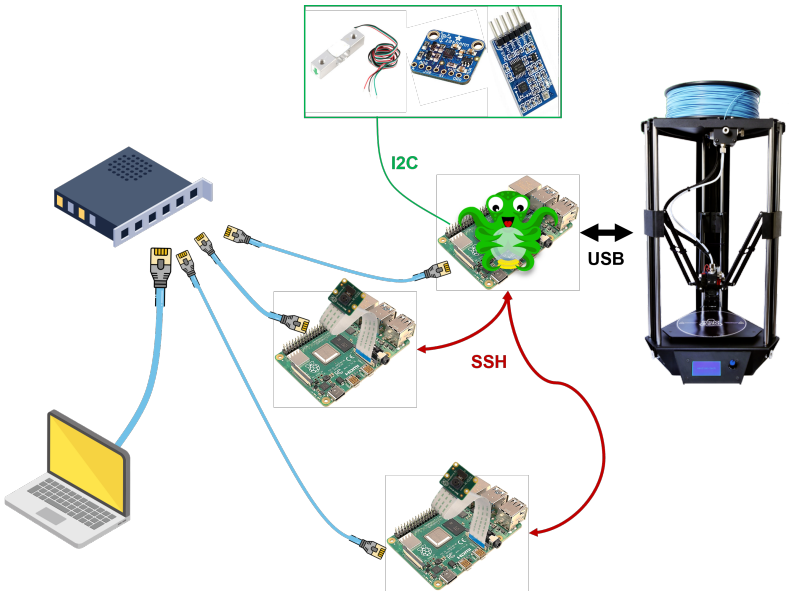


FIGURE 2.10: Sensor Linkage Through Local Network

Sensor	Signal	Unit
Printer integrated sensors	Target nozzle temperature	°C
	Actual nozzle temperature	°C
	Target board temperature	°C
	Actual board temperature	°C
	Layer number	
IMU	Nozzle acceleration	$m \cdot s^{-2}$
	Nozzle angular speed	$rad \cdot s^{-1}$
	Nozzle exterior temperature	°C
	Nozzle magnetization	$A \cdot m^{-1}$
Board accelerometer	Board acceleration	$m \cdot s^{-2}$
Weight	Wire spool weight	$kg \cdot m \cdot s^{-2}$

TABLE 2.4: Signals Measured on the 3D Printer

2.3.4 Data Collection

In all machine learning approaches, building a relevant dataset is key. The quality of data is often more important than any kind of tuning done on the hyper-parameters. This is because the key information (the signal signatures that precede a fault) that the algorithm wants to detect needs to be present in the data, otherwise it cannot be detected. However, in our case, as mentioned previously, we do not fully know how the system reacts to some parameters or combinations of parameters, and so we do not know in which signals the fault signatures lie. Thus, we want to get as many signals as input as possible and with the maximum level of accuracy possible. Still, computation time can quickly become an issue (especially in real-time contexts). This is part of the reason why cameras have not been considered (see Appendix A on page 113), they often require heavy image processing.

All the signals gathered and preprocessed to be fed to the algorithms are recapped in Table 2.4.

The nozzle magnetization is available in the IMU but is not expected to give meaningful data.

FIGURE 2.11: Labeling Tool

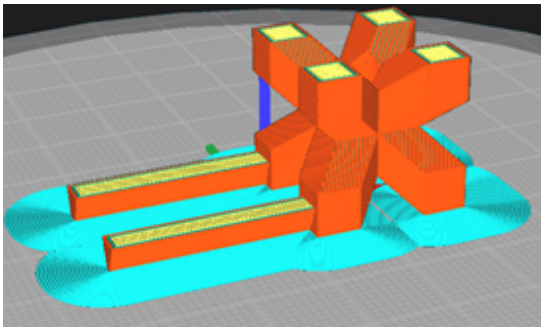
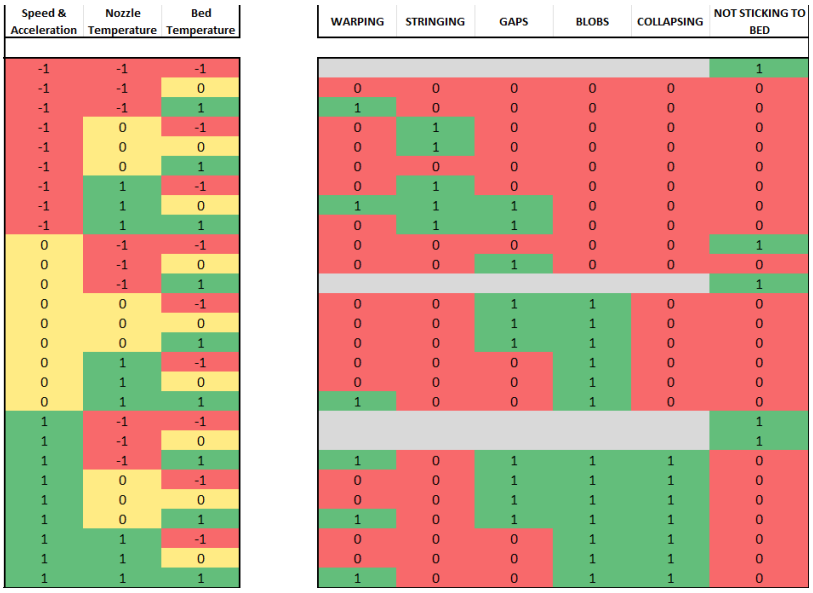


FIGURE 2.12: Geometry Designed to be Able to Generate all 8 Studied Types of Faults

This data needs to be labeled with the occurring fault in order to train supervised machine learning algorithms. In order to know which fault occurs during each time frame, we recorded it during printing using a synchronized chronometer. This makes 8 time-series (one for each type of fault) of zeros (nominal) and ones (faulty) that last as long as the print lasts. In practice, the timings where faults occurred were written down during printing and then turned into time-series using a homemade interactive interface to directly add them to the rest of the dataset. This interactive interface allows displaying an input signal of choice on a graph, and selecting portions of the print to declare faults that occurred in said portions by clicking on the appropriate fault button (see Figure 2.11). Using this tool, it is also possible to declare when the print starts in order to get rid of the section where the printer is still being initialized (the analysis is done on data of parts being actually printed). An experimental design was imagined in order to know which parts to print with which parameters for the dataset. It ensures faults appear with a high enough frequency to get interesting training data without biasing the dataset. The diversity comes from initial parameters, either slicing parameters or the geometry of the printed part. For the experimental design, we mainly used one geometry specifically designed to generate as many different types of faults as possible while keeping a relatively simple shape to be able to print it in a short time (see Figure 2.12). It takes between 10 and 40 minutes to print this part (depending on printing speed). It was 25 minutes for standard parameters. However, we soon realized that more diversity in the geometrical shape was required and decided to use five more shapes. A total of 54 parts were printed



Speed & Acceleration	Nozzle Temperature	Bed Temperature	WARPING	STRINGING	GAPS	BLOBS	COLLAPSING	NOT STICKING TO BED
-1	-1	-1						1
-1	-1	0	0	0	0	0	0	0
-1	-1	1	1	0	0	0	0	0
-1	0	-1	0	1	0	0	0	0
-1	0	0	0	1	0	0	0	0
-1	0	1	0	0	0	0	0	0
-1	1	-1	0	1	0	0	0	0
-1	1	0	1	1	1	0	0	0
-1	1	1	0	1	1	0	0	0
0	-1	-1	0	0	0	0	0	1
0	-1	0	0	0	1	0	0	0
0	-1	1						1
0	0	-1	0	0	1	1	0	0
0	0	0	0	0	1	1	0	0
0	0	1	0	0	1	1	0	0
0	1	-1	0	0	0	1	0	0
0	1	0	0	0	0	1	0	0
0	1	1	1	0	0	1	0	0
1	-1	-1						1
1	-1	0						1
1	-1	1	1	0	1	1	1	0
1	0	-1	0	0	1	1	1	0
1	0	0	0	0	1	1	1	0
1	0	1	1	0	1	1	1	0
1	1	-1	0	0	0	1	1	0
1	1	0	0	0	0	1	1	0
1	1	1	1	0	0	1	1	0

FIGURE 2.13: Experimental Design and the Fault Observed During each Print

using 6 different shapes and various printing parameters. Based on experts' feedback, three parameters are known to impact more significantly the quality of parts than others:

- nozzle temperature
- bed temperature
- printing speed

To make sure to cover their possible field of values, we took the slicer most extreme possible values. We used the slicer Cura (Šljivic et al., 2019). The first 27 parts of the dataset are printed using these extreme parameters and also the default values (default in the slicer). The experimental design is represented on Figure 2.13 where -1 and 1 respectively represent the lower and upper bounds for these input parameters, 0 their default value, on the left hand side, whereas 0 means not present and 1 means present for the faults on the right hand side. In the experimental design, the faults *Layer Shifting* and *Not Extruding* were not considered because of their very low and often random presence. Also, the fault *No Adhesion* (or *Not Sticking to Bed*) causes the print to stop, so no other later fault could be observed on these prints. The other parts for the dataset were printed with other various shapes and trying to generate faults that appeared less in the experimental design. However, some fault such as *Layer Shifting* only appear occasionally (they do not remain over time) so they inherently end up with less samples.

2.3.5 Data Preprocessing

Preprocessing is a crucial part of machine learning because it allows to extract meaning from the data and make it more obvious for the algorithms to learn it.

First of all, all signals are synchronized on the highest frequency signal (the board accelerometer frequency). Its period is $3e^{-3}$s. Empty values are filled using linear interpolation (Lepot, Aubin, and Clemens, 2017).

Once the signals and labels are gathered in the same table, we extract sliding windows of a fixed amount of seconds from the time-series. For this experiment, we chose windows of 10 seconds which corresponds to 3330 time-steps, with a stride half the size of the window, meaning two successive windows share half of their data.

For each print, we end up with a 3-dimensional table with dimensions:

$$(n_{windows}, 3330, 39)$$

where $n_{windows} = \frac{PT}{10}$ with PT the printing time in seconds and 39 is 38 observable variables and 1 label.

The same process with windows of 1s and windows of 50s was implemented. Multiple sizes are tested because, again, the behavior of the fault signatures is not accurately known. With a stride of half the window size, a 1s window allows the algorithm to compute for 0.5s. Less than that would be too much of a constraint for a machine learning algorithm. Meanwhile, a 50s window should identify most signatures (once again, according to 3D printing experts). The 10s window is a good trade-off and ended-up having the best results.

Gathering time-steps in windows allows us to detect patterns on various time ranges. Indeed we expect the faults to have a small signature in the signal. Still, we do not have prior knowledge on the size or behavior of this pattern, hence the different window size trials.

Once the windows are extracted, they must be labeled. It was decided to label every window as faulty (1) or normal (0) according to each type of fault. Each algorithm gets fed the whole window and is trained to output 0 or 1 for its corresponding faults. For critical faults, the presence of a faulty time-step in the window (even only one) results in the window being labeled as faulty. Meanwhile, for severe faults, if the majority of the time-steps are faulty the window is labeled as faulty for this fault, otherwise it is nominal.

$$l_{CriticalFault} = max_{i \in \text{window}}(l_i) \tag{2.1}$$

$$l_{SevereFault} = \begin{cases} 1 & \text{if } mean_{i \in \text{window}}(l_i) \geq 0.5 \\ 0 & \text{if } mean_{i \in \text{window}}(l_i) < 0.5 \end{cases} \tag{2.2}$$

where l_* is the label of any window w.r.t. fault `"*"` and l_i is the label of the i^{th} time-step within the same window.

Let us note that this labeling allows detection of faults but not anticipation. To be able to train the algorithms to *predict* faults, the label is shifted one window backwards. It means that a window is labeled with a fault that will take place in the next window.

2.3.6 Feature Engineering

In order to make data intelligible to algorithms, feature engineering is performed on that data. The list of features extracted from every window is shown in Table 2.5 on the next page. To enhance algorithm performances and inference time, feature selection is also performed before training each algorithm using scikit-learn. The windows, now characterized by the features, are split 80%/20% between a training set and a testing set. Because algorithm performances can greatly vary according to the data put in each set, two means of splitting the dataset are used:

- **Statistical split**: separate the windows between training and testing randomly and regardless of which print they belong to.

Name	Signification
mean	Mean value
std	Standard deviation
mad	Median absolute deviation
max	Largest value in array
min	Smallest value in array
iqr	Interquartile range
maxPeak	Largest frequency value
meanFreq	Frequency signal weighted average

TABLE 2.5: Features Engineered in each Window

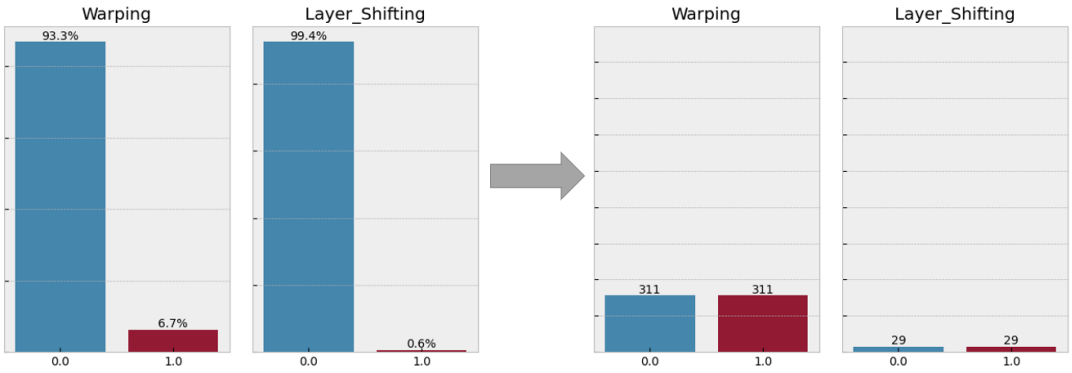


FIGURE 2.14: Balancing the Dataset for each Fault Type

- **Objective split**: separate the prints between training and testing and then extract the windows. This means that two windows from the same print can not be in different sets.

Once the splits are done, within each set and for each fault type separately, the classes are balanced so the algorithms do not get biased. The goal is to use the machine learning algorithms as binary classifiers. Thus, 8 versions of each algorithm are trained in order to predict the presence of each fault type. This allows for multi fault prediction while still using only binary algorithms. However, it is more computationally expensive.

More often than not, there are more nominal than faulty windows so this means removing some nominal windows. We used a *One-versus-Rest* (See Tax and Duin, 2002) method to create our balanced datasets. For each fault, this means a dataset with half the windows containing said fault, and the other half any window that does not contain the fault (it can be any combination of nominal or the other faults). An example is presented in Figure 2.14.

2.3.7 Training

Once the data is balanced, it can be used to train algorithms.

In order to obtain relevant results, a 10-Folds Cross Validation is applied on the training sets to assess how the results of the training generalize to a new set of prints.

For each of the eight fault types, six machine learning algorithms were trained (see Section 2.1.2 on page 8). These algorithms and the value of their hyper-parameters are presented in Table 2.6 on the facing page. They are all implemented using the scikit-learn library for Python. The hyper-parameters descriptions can be found in

Algorithms	Hyper Parameters	Values
Logistic Regression	C solver multiclass penalty	$\in]0, 5]$ "liblinear" "auto" "l1"
Linear Discriminant Analysis	$n_{components}$	100
K-Nearest Neighbors	$n_{neighbors}$	3
Random Forest	$n_{estimators}$	400
Decision Tree	max_depth	None
Multi-layer Perceptron	hdn_layer_sizes activation	(64,32,16,) "tanh"

TABLE 2.6: Algorithms and Hyper-parameters

	LR		LDA		RF	
	Mean	STD	Mean	STD	Mean	STD
Warping	88.47	1.01	93.3	0.61	99.85	0.12
Layer Shifting	69.85	8.22	67.43	6.54	90.99	4.67
Blobs	72.67	0.66	93.14	0.28	99.79	0.08
No Adhesion	85.40	2.30	96.41	1.35	99.46	0.42
Stringing	73.6	0.31	95.29	0.26	99.94	0.04
Gaps	77.67	0.46	96.53	0.22	99.71	0.09
Overheating	60.6	0.69	91.94	0.28	99.94	0.03
Not Extruding	70.09	2.16	92.31	0.7	99.22	0.35

TABLE 2.7: Results for the Statistical Split (1/2)

the scikit-learn documentation. The tuning of these hyper-parameters is done by performing a grid search in commonly used intervals or given by expert knowledge.

2.3.8 Results

The performance of these algorithms on windows of 10 second is shown in Table 2.7 and 2.8 on the following page for the statistical split and in Table 2.9 on the next page and 2.10 on page 27 for the objective split. with *Mean* being the average accuracy on each validation of the cross-validation on the testing set and *STD* the standard deviation. The best scores are colored. Green is used when the score is above 90% and yellow otherwise. In the statistical split, the Random Forest algorithm is outperforming other models by a large margin, and as expected, models trained on the statistical split are always better than the ones trained on the objective split.

On Figure 2.15 on page 27, a decision tree trained to predict the fault *No Adhesion* is presented. The first criterion to discriminate between faulty and nominal is whether the bed temperature is above or below 31°C. Indeed, with a bed (or board) temperature too low, the printed part does not stick to the bed. Decision trees are not only one of the best performing algorithms but also present a level of explainability that no other algorithm tested here do.

	DT		MLP		KNN	
	Mean	STD	Mean	STD	Mean	STD
Warping	98.91	0.36	67.65	11.85	88.4	0.82
Layer Shifting	84.46	4.99	69.44	6.36	72.99	7
Blobs	99.19	0.19	53.75	5.75	80.49	0.54
No Adhesion	98.15	0.50	73.76	8.91	81.21	2.08
Stringing	99.84	0.04	52.81	1.92	84.13	0.33
Gaps	99.67	0.14	53.56	3.11	78.83	0.81
Overheating	99.73	0.06	53.47	1.23	81.28	0.31
Not Extruding	97.59	0.44	65.51	8.74	84.93	0.83

TABLE 2.8: Results for the Statistical Split (2/2)

	LR		LDA		RF	
	Mean	STD	Mean	STD	Mean	STD
Warping	67.24	13.83	51.43	17.67	59.10	13.20
Layer Shifting	50.4	25.57	53.67	24.11	48.03	11.22
Blobs	61.49	8.96	40.82	9.59	49.25	3.48
No Adhesion	55.77	16.98	46.49	5.73	50.49	2.48
Stringing	66.42	16.93	59.60	15.35	65.64	16.59
Gaps	61.10	17.23	59.24	21.14	54.23	5.11
Overheating	47.85	16.59	57.24	15.85	58.7	14.32
Not Extruding	59.29	14.2	65.75	15.78	73.58	22.97

TABLE 2.9: Results for the Objective Split (1/2)

2.3.9 Takeaways

In this section, we analyze the results and what should be done to improve the method.

Split and Increase the Dataset In Section 2.3.6 on page 23 we mentioned the two manners to split the dataset. The objective split is relevant to our application. Indeed, we want the algorithms to be able to predict faults on new prints that do not already belong to the dataset. The results highlight the advantages of using different algorithms for different faults. Indeed, there is not a single algorithm having the best performance on all faults. However, the performances in the objective split are lackluster because some faults are not represented in the training dataset. Indeed, some faults appear in only a few prints, meaning that if those prints are all in the testing set, the algorithms are not trained on them and thus not able to predict them. On the other hand, if they are all in the training set, the algorithm is made to predict them correctly but it is not checked by the testing set. As a result, despite the objective split being more suitable for the goal, its performance is limited by the way the dataset is built. Thus, we used the statistical split based on the assumption that the results of this split are representative of a case where all possible print scenarios are present in the dataset. Indeed, if all possible print scenarios are present in the training dataset, that means that data in the testing dataset will be similar to the training data, just as in the statistical split case.

If more parts are printed and included in the dataset, the case where a fault type is not present in either the training set or the testing set is less likely to happen. Also, the main consequence of choosing one splitting method over the other is the different degree of similarity between the training and testing sets. With a large enough dataset, this difference disappears and the performance should also become

	DT		MLP		KNN	
	Mean	STD	Mean	STD	Mean	STD
Warping	57.84	15.70	56.78	11.95	53.35	11.31
Layer Shifting	54.72	20.80	49.41	19.97	47.11	18.36
Blobs	47.15	6.43	52.74	4.21	48.39	7.71
No Adhesion	52.51	6.9	56.92	15.58	53.41	13.36
Stringing	63.38	16.11	42.66	10.36	53.61	9.41
Gaps	56.69	9.83	42.57	9.43	47.75	9.4
Overheating	59.69	14.85	45.45	11.35	46.64	7.79
Not Extruding	68.02	18.70	53.40	15.56	56.38	12.01

TABLE 2.10: Results for the Objective Split (2/2)

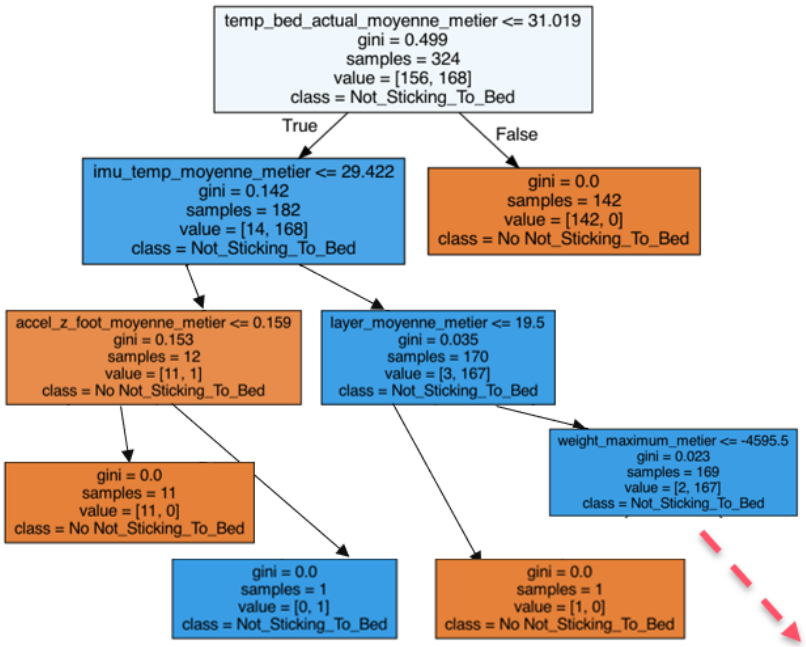


FIGURE 2.15: Decision Tree Trained to Predict the *No Adhesion* Fault (Blue=Faulty, Orange=This fault is not present)

equivalent. This is why using the statistical split gives an idea of how good machine learning algorithms could be. Still, no proof is established that this is true so the results from the statistical split should be taken with a grain of salt. In a more generic sense, improving the dataset should definitely improve the performance. In particular, the focus has to be on increasing the presence of rare faults and increase the number of prints they appear in. Also, increasing the diversity of part shapes is important to increase the representativity of the dataset.

Geometrical Analysis The 3D printing experts all agree on the fact that the geometrical shape of the part to be printed is a very important factor for faults. In this experiment, the dataset was built using 6 different shapes for its parts. A geometrical shape is a combination of many basic shapes (such as pyramids, cylinders and whatnot). Perhaps, a geometrical study could help prove that all 3D parts are made of a finite set of elementary shapes and this elementary shapes could be used to train the fault prediction algorithm, thus increasing its robustness to new, unknown shapes.

Computer Vision In order to identify anomalies, the printer has been equipped

with two cameras. There is a possibility that these cameras can detect precursors to specific types of fault. The reason we think it is possible is that it actually happens on the printer we use at Atos. At the beginning of each print, when the nozzle heats up before the print, a small blob of PLA forms at the tip of the nozzle. When the print starts, it can remain embedded in the part and cause a blob on it. Some works such as Delli and Chang, 2018 are able to detect faults when they occur, in this case using Support Vector Machines. However, once the fault happens, it is too late. Henceforth, we have to detect precursors. Since we do not know what form those precursors could take, it would require either to compare the current printed part with its 3D model (maybe with a geometrical analysis similar to what the authors in Petsiuk and Pearce, 2020 did) or to train a computer vision algorithm to predict the fault using labels from our already built dataset.

Hybrid Methods It may be that, even if the amount of data, its precision, and its quality are increased, the algorithms are not able to predict the faults. A possible reason for that would be because the algorithm structures are not able to represent the real system behavior. This goes back all the way to the fact that the mathematical equations that govern the behavior of the system are not fully known. This prevents the use of a model-based method. However, nowadays, many hybrid methods for diagnosis have been developed to compensate for the lack of knowledge about the system with data. The first step is to estimate some system equations using machine learning (for instance with symbolic regression, as explored in the next chapter). An advantage of most hybrid methods over pure data-based methods is explainability of the outcome. In order to eventually correct the faults in real-time, explainability is a must-have.

2.4 Conclusions

This chapter introduces various machine learning concepts that are used throughout the manuscript. It also presents a state of the art of 3D printing diagnosis methods that shows that almost all approaches solely rely on data-driven techniques.

Afterwards, machine learning is applied to our 3D printer use case. Applying data-driven methods to the 3D printer proved challenging for one main reason: it is very complex to build a relevant dataset. However, some algorithms showed decent performances and, in particular, decision trees managed to identify interesting patterns in the variables that could be linked to expert knowledge about the system.

Machine learning algorithms learn underlying patterns in the data. Those pattern rarely have a physical reality in the system. Identifying patterns with a physical sense would increase performances but also give some knowledge about the studied system. This leads to the idea of constraining data driven algorithms to learn something about the system rather than to learn arbitrary relations between the variables. This is explored in Chapter 4 on page 45. The next chapter focuses on diagnosis methods that try to combine model-based and data-driven approaches.

Chapter 3

Hybrid AI Diagnosis

Considering the results obtained in the previous chapter, we look for methods that combine model-based and data-based diagnosis. The main idea is to look at structural diagnosis methods that rely on the structure of the model of the system rather than on the analytical equations. This allow to work with non-linear, dynamical, high dimension systems. Thus, this chapter starts by providing background on structural analysis. It is a powerful model-based diagnosis tool that can be used in conjunction with data-driven approaches. Then, we give an overview of state-of-the-art diagnosis methods applied to dynamic systems, focusing on methods that combine data-driven and model-based approaches. After that, we propose a new method that enhances such an existing method, resulting in a combined data-driven and model-based diagnosis method. The proposed method is tested on the water tanks use case, which is also introduced. The goal is to study which approaches already exist, where are their shortcomings and how to build upon what exists in order to overcome the hurdles that were faced.

3.1 Background - Structural Analysis

3.1.1 Important notions

Before talking about structural analysis, we need to define the notion of residual generator for a system Σ, as defined in Section 1.4.1 on page 3.

Definition 23 (Residual Generator for Σ)**.** *A residual generator for $\Sigma(\mathbf{z}, \mathbf{x}, \mathbf{f})$ is a relation $arr(x', \dot{x}', \ddot{x}', \ldots) = r$ (with x' a sub-vector of $\mathbf{x}$[1] and r a scalar named residual) such that for all x consistent with $\Sigma(\mathbf{z}, \mathbf{x}, \mathbf{f})$, it holds that in steady state $r = 0$.*

A relation $arr(x', \dot{x}', \ddot{x}', ...) = r$ as defined in Definition 23 is called an Analytical Redundancy Relation (ARR). An ARR is sensitive to faults in the system. Indeed, deviating from a nominal case leads to $r \neq 0$. However, not all $f \in \mathbf{f}$ necessarily appear in the expression of an ARR.

Definition 24 (Fault Support)**.** *The fault support FS of an ARR is defined as the set of faults that impact this ARR. $FS \subset C$.*

Thus, when the residual of an ARR is non-zero, it means that at least one of the faults of the fault support of this ARR has occurred.

Definition 25 (Signature Matrix)**.** *A signature matrix is a matrix where a line i corresponds to ARR_i, a column j corresponds to the fault f_j, and a cell c_{ij} with a 1*

[1] Reminder that $\mathbf{x}$ represents observable variables contrary to what is usually done in diagnosis.

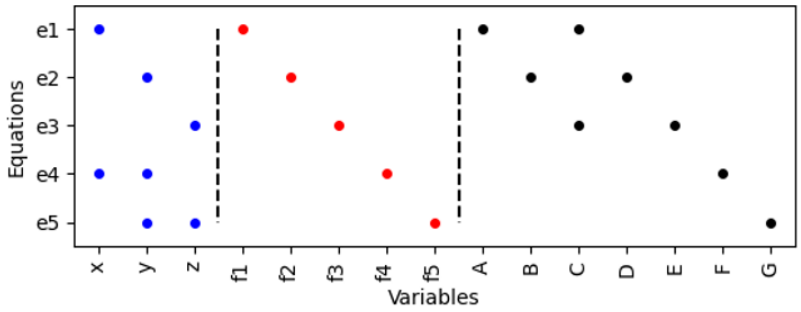


FIGURE 3.1: Example of a Structural Model. Each equation corresponds to a component of the system. The presence of a dot signifies that the variables belongs to the equations, the absence that is does not. The left-most section shows non-observable variables, the central section shows faults and the right-most section shows observable variables.

means that the fault belongs to the fault support FS_i of ARR_i.

$$c_{ij} = \begin{cases} 1 \text{ if } f_j \in FS_i \\ 0 \text{ if } f_j \notin FS_i \end{cases} \tag{3.1}$$

A fault f_j is said to be structurally detectable if and only if column j of the signature matrix contains at least a 1. A fault f_j is isolable if and only if $\neg\exists\ j', j' = j$, in other words f_j is isolable if and only if all other columns are different from j. See Definitions 9 on page 5 and 11 on page 5.

Structural analysis is a general framework that can be used to analyze large-scale, complex and dynamic systems described by numerous equations, both linear and non-linear. It abstracts equations by only keeping their links with variables. Therefore, it ignores the details of parameter values to base the analysis on the structure of the system by means of efficient graph-based tools (Cassar and Staroswiecki, 1997) and thus a major advantage of structural analysis is that it can be used for systems under uncertainty for which the analytical model is not precisely known (Cassar and Staroswiecki, 1997; Düştegör et al., 2006).

The *structural model* M of a system represents this system with its components and the constraints related to these components. It can be obtained by abstracting the functional relations of $\Sigma(\mathbf{z}, \mathbf{x}, \mathbf{f})$.

The structural model M can be represented by a matrix qualified as the *incidence matrix,* which rows are associated to equations and columns to variables. Its elements take the value 1 when the variable is involved in the equation and 0 otherwise. An example of such a representation is given in Figure 3.1. For instance, in this illustration, equation e_1 includes the unknown variable x, the fault f_1 and the observable variables A and C.

Equivalently, M can be represented by a *bipartite graph* $G(\Sigma \cup \mathbf{x} \cup \mathbf{z}, A)$, where A is the set of edges linking equations of $\Sigma(\mathbf{z}, \mathbf{x}, \mathbf{f})$ and variables of $\mathbf{x}$ and $\mathbf{z}$. Hence, each edge links a variable with an equation it belongs to.

3.1.2 Diagnosis via Structural Redundancy

When used for fault detection and isolation purposes, structural analysis aims at finding subsets of system equations with redundancy. These can be turned into

diagnosis tests, i.e. ARRs or parity relations, which are designed off-line (Blanke et al., 2006). Diagnosis tests are then checked against observations on-line.

Redundancy in a system of the form $\Sigma(\mathbf{z}, \mathbf{x}, \mathbf{f})$ can be brought to light by the well-known Dulmage-Mendelsohn (DM) canonical decomposition (Blanke et al., 2006; Murota, 2009; Dulmage and Mendelsohn, 1958). It partitions the system into three subsystems:

- Σ^+ has more equations than unknown variables and is named the *structurally overdetermined* (SO) subsystem,
- Σ^0 is the *structurally just determined* subsystem,
- Σ^- has more unknown variables than equations and is named the *structurally underdetermined* subsystem.

If a set of equations Σ' is such that $\Sigma' = \Sigma^+$ and no proper subset of Σ' is overdetermined, this set Σ' is qualified as *minimally structurally overdetermined* (MSO) (M. Krysander, Åslund, and E. Frisk, 2010). This means that an MSO set has exactly one more equation than unknown variables, which is a particular case of SO subsystems. Nevertheless, only MSO sets impacted by faults are interesting for diagnosis. This is why the concept of fault support was defined further up.

Definition 26 (Fault-Driven Minimal Structurally Overdetermined Set)**.** *A Fault-Driven Minimal Structurally Overdetermined (FMSO) set is an MSO set whose fault support is not empty (Pérez et al., 2017).*

Definition 27 (Structural Redundancy)**.** *The structural redundancy $\rho_{\Sigma'}$ of a set of equations $\Sigma' \subseteq \Sigma$ is defined as the difference between the number of equations and the number of unknown variables in Σ'.*

If a set of equations is structurally redundant ($\rho_{\Sigma'} > 0$), it means that residuals can be generated using the equations in this set. Once the subsets of equations with redundancy are found using structural analysis, several methods exist to find the analytical expression of the residual generator (L. Travé-Massuyès, Escobet, and Olive, 2006; Chow and Willsky, 1984; E. Frisk and M. Nyberg, 2001).

An FMSO set φ identifies a just overdetermined subset of $|\varphi|$ equations of the model, among which, one is redundant. This means that all the unknown variables can be determined using $|\varphi| - 1$ equations, and that an ARR can be generated by substituting in the $|\varphi|^{th}$ equation. This residual generator can then be used to diagnose faults in its fault support.

Structural analysis is illustrated on an example in Section 5.1.1.3 on page 70.

3.2 State of the Art

As mentioned in the introduction, diagnosis can be divided into two main subfields, model-based diagnosis and data-driven diagnosis (Q. Yang, 2004). The main goal of this thesis is to research how to combine these two fields in order to take advantage of both and mitigate their limitations.

Next sections define what we call *Hybrid AI diagnosis methods*, then focus on diagnosis methods that specifically combine machine learning and structural analysis.

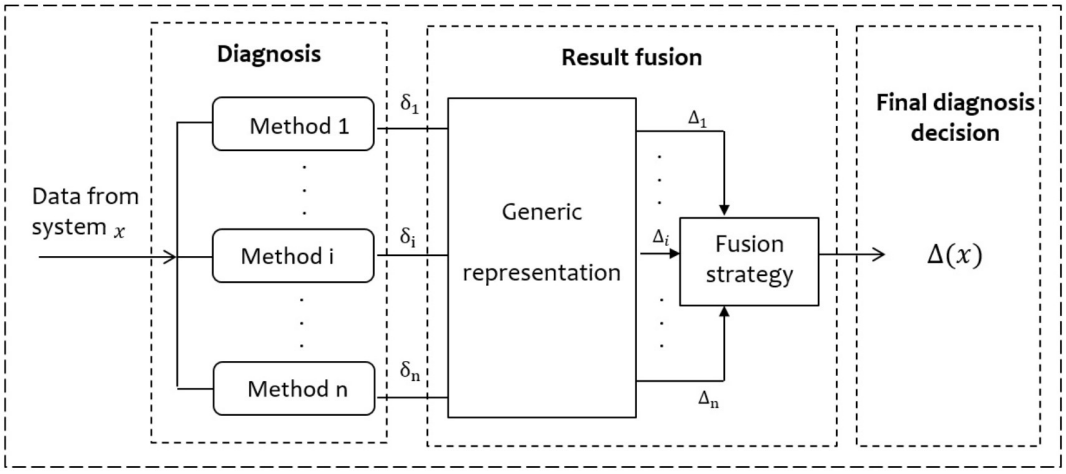


FIGURE 3.2: Principle of Fusion

3.2.1 Hybrid AI Diagnosis Methods

Definition 28 (Hybrid AI Diagnosis Method)**.** *We call hybrid AI diagnosis method a diagnosis method that combines model-based diagnosis and data-driven diagnosis.*

In general, hybrid AI diagnosis methods use artificial intelligence based methods such as machine learning, hence the name.

While establishing a state of the art of hybrid AI diagnosis methods, two main families have been identified:

- Fusion and similar methods that consist in using both model-based and data-driven approaches side-by-side and then combining the outputs into one final prediction.
- Model identification using machine learning in order to then use model-based methods.

Fusion An example of what fusion can look like in a generic context is given in Figure 3.2 that comes from Slimani et al., 2018. Slimani et al., 2018 propose a framework to use any kind of method, no matter the inputs and outputs, using a standardized representation for data, either collected from the system, or in the shape of model-based knowledge. Nevertheless, most diagnosis fusion strategies rely on either fully model-based methods or fully data-driven methods (S. Li et al., 2020, Duan et al., 2018, Shao et al., 2021)

Model Identification Many works provide system identification (Keesman, 2011, Ljung, 1995, Åström and Eykhoff, 1971) or model identification (Hollerbach, Khalil, and Gautier, 2016, Janos Abonyi and János Abonyi, 2003, K. Li, Peng, and Irwin, 2005) strategies. However, only a few use that to then perform diagnosis of the system using model-based methods. Subias and Travé-Massuyes, 2006 propose such an approach that consists in a discretization algorithm that extracts a qualitative model of the system from data. This model is then used to perform model-based diagnosis.

Other Methods It is worth mentioning that some data-driven methods, while not hybrid, focus on explanation of a diagnosis (which is part of what we are looking for in hybrid AI diagnosis methods). For instance, Basak and Krishnapuram, 2005 provide an unsupervised decision tree algorithm that outputs insights as to why a fault occurs. A lot of works focus on extracting knowledge from black-box models

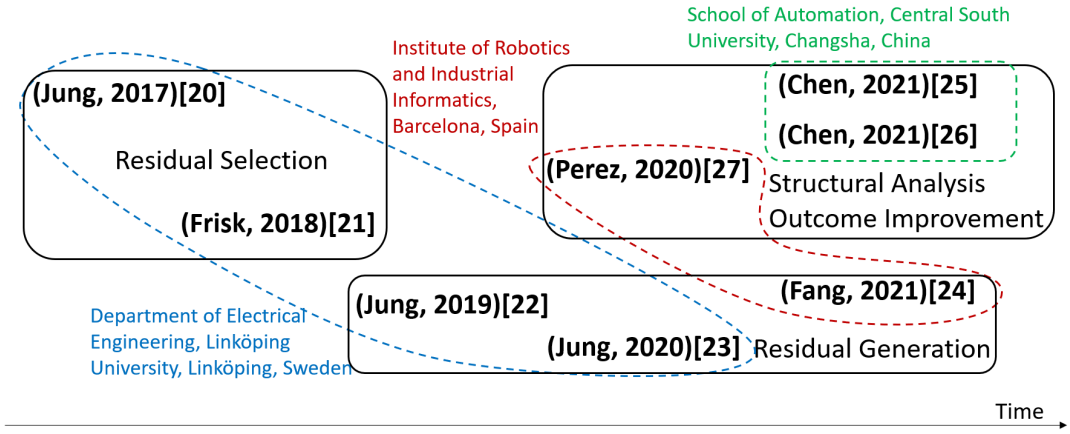


FIGURE 3.3: Summary of Methods Combining Structural Analysis and Machine Learning.

such as neural networks (Rudin, 2019, Zerilli, 2022, Mothilal, Amit Sharma, and Tan, 2020, Gilpin et al., 2018).

On the other hand, some model-based methods take inspiration from machine learning architectures, for instance Console, Picardi, and Duprè, 2003 propose to build a temporal decision tree with the model of the system and use it to perform diagnosis.

Following this state of the art, a subfield of model-based diagnosis methods has been identified as promising. Indeed, structural analysis relies on structural knowledge about the system and not on physical knowledge. This can be taken advantage of.

3.2.2 Machine learning and structural analysis

Our paper Goupil, Chanthery, et al., 2022 provides a review of existing methods combining machine learning and structural analysis. Figure 3.3 summarizes the findings. Three main approaches have been identified:

- **Residual Selection**: the step of structural analysis that consists in selecting the right expression for generating a residual is replaced with a machine learning algorithm. For instance, D. Jung and Sundstrom, 2017 replace this step by a feature selection algorithm (see Figure 3.4 on the following page).
- **Residual Generation**: the evaluation of a residual in structural analysis requires the equation of the model. Thus, some works propose to replace this step with a machine learning algorithm trained with labeled data. For instance, D. Jung, 2020 replaces residual generation with a grey-box recurrent neural network (Figure 3.5 on the next page). We explore and extend this possibility further in Section 3.3 on page 35.
- **Improvement post structural analysis**: three papers try to improve the output results of structural analysis with deep learning. This does not reduce the requirements for using the model-based framework but enhances the diagnosis accuracy by taking advantage of available data. For instance, Z. Chen et al., 2021 uses graph convolutional networks to enhance the diagnosis accuracy of structural analysis (Figure 3.6 on the next page).

From this study, the conclusions are:

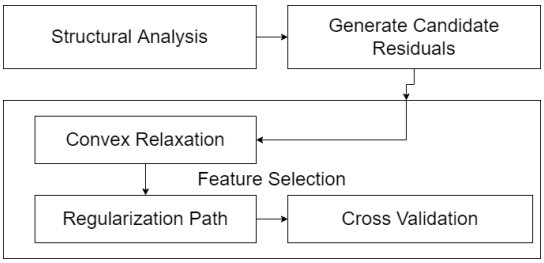


FIGURE 3.4: Replacing Residual Selection with a Feature Selection Algorithm

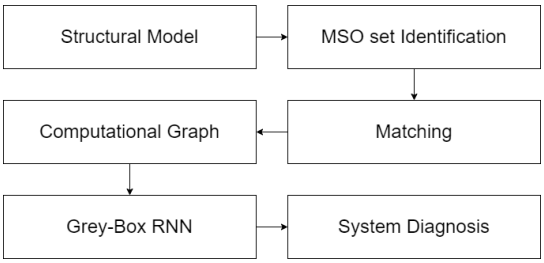


FIGURE 3.5: Replacing Residual Generation with a Grey-Box Recurrent Neural Network

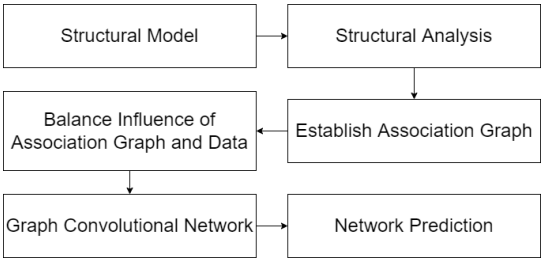


FIGURE 3.6: Improving Structural Analysis Results with a Graph Neural Network

- These various approaches can be used together to enhance accuracy and reduce the need for model knowledge.
- Some works (Svärd et al., 2011, for instance) utilize machine learning for residual computation (outside the scope of structural analysis). This could be put to good use.
- Here, the focus is on residuals. Could machine learning be used in other areas of structural analysis ? For instance, automatically generating the structural model using graph neural networks would lower the model knowledge requirements of structural analysis.

Finally, on a positive note, those methods rarely require large amounts of data. In general, they require some varied data that cover all types of faults. This shows that combining some knowledge about the system and some data can output convincing results. This is promising because more often than not, some knowledge of the system is available: expert knowledge, basic components equations, etc. Also, some data is often available, such as nominal data and a few faulty occurrences.

3.3 Variation on an hybrid AI Diagnosis method

The above state of the art shows that no algorithm intrinsically combines model-based diagnosis and data-based diagnosis. Proposing such a method is the focus of the next chapter. Papers such as D. Jung, 2020 use a model-based algorithm but replace a step of this algorithm with a data-based method. Still, this approach is very promising and shows that structural analysis is a good tool for hybrid AI diagnosis methods.

This section proposes a hybrid AI diagnosis method. This method is heavily inspired by D. Jung, 2020 which combines structural analysis with recurrent neural networks. It replaces the residual computation (which requires exact physical equations of the system) with a recurrent neural network trained to act as the residual, meaning to output 0 for data from a class that is not supposed to be detected by the residual and not 0 otherwise.

3.3.1 The SA-ML Method

In our case, residual computation is achieved by replacing recurrent neural networks with any type of machine learning algorithm. So the method is called SA-ML as "Structural Analysis and Machine Learning". Furthermore, for each residual, the best algorithm is kept and different residuals can have different algorithms used to simulate them. Another major difference with D. Jung, 2020 is that dynamics of the system are considered in a very different way. They use recurrent neural networks to deal with time-series signals as a whole, using the information of the order of the timesteps. We use derivatives that inherently contain information about variations of the variables.

The principle of the method is described in Figure 3.7 on the following page. All the steps are detailed in the following sections when applied to the water tanks system.

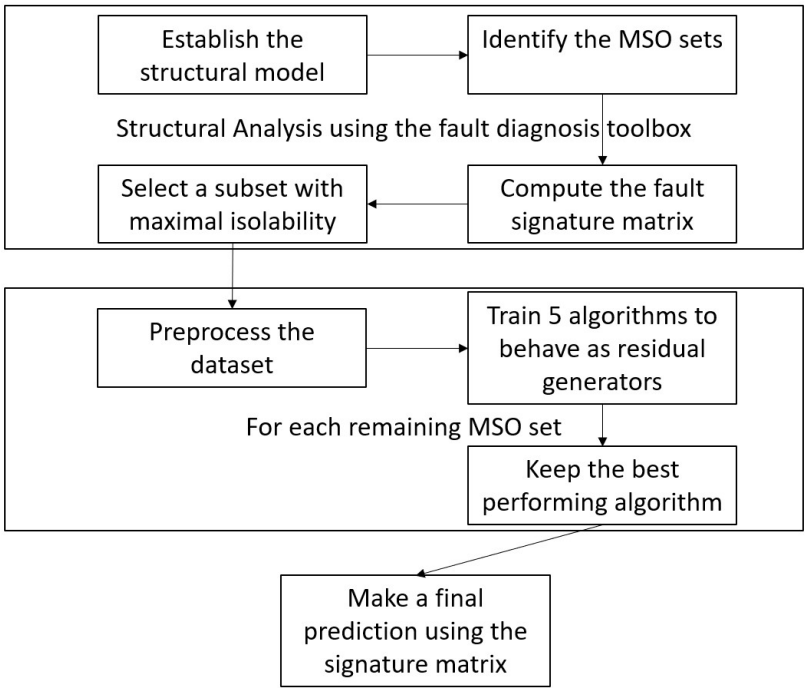


FIGURE 3.7: Summary of the Proposed Method

3.4 Application to a dynamic non-linear system

3.4.1 System description: the two tanks system

The studied non-linear, dynamic system consists of two coupled water tanks. Faults considered in the process are leakages and actuator and sensor faults. Each tank is equipped with a level sensor with values y_1 and y_2. The flow out of each tank is measured with a flow sensor with values y_3 and y_4. The system is presented in Figure 3.8 on the next page. The tanks are subject to ten possible faults, listed in Table 3.1 on the facing page. Here, F_a denotes fault mode and f_a denotes the modeled fault, e.g., the fault signal. The different fault modes that can be introduced in the system can be modeled in several different ways, e.g., as signals or deviations in constant parameters. Actuator and sensor faults are modeled using additive signals, and other fault modes as constant parameters. Data for this system is generated in Python using `odeint` from the scipy package. The input flow of $\mathbf{T}_1$, u_{ref}, in function of time t^*, is described by Equation 3.2:

$$u_{ref}(t^*) = \begin{cases} 3 \text{ if } 0 \leq t^* < 1 \\ 5 \text{ if } 1 \leq t^* < 15 \\ 6 \text{ if } 15 \leq t^* \end{cases} \tag{3.2}$$

Its graph is shown in Figure 3.9 on page 38. Simulations are 700s long, chosen in order to be able to see the impact of the fault on the measurements. Faults randomly occur between 15s and 300s. Data is labeled as faulty from the fault occurrence time. Before, or on simulations without fault, it is labeled as nominal. We only consider cases where zero or one fault occurs, not multiple faults at the same time. The amplitude of fault varies according to fault type, and is randomly selected in the intervals given in Table 3.1 on the next page. When a fault occurs, it ramps up for two seconds before reaching its full amplitude. An example of such a fault is shown

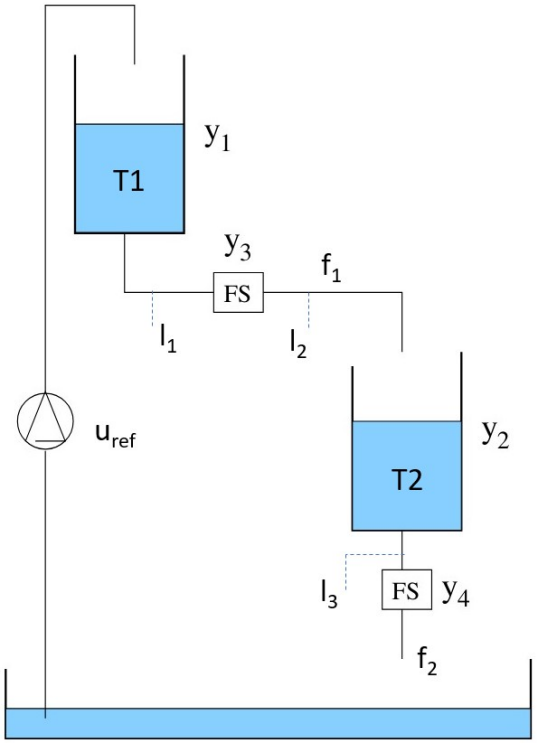


FIGURE 3.8: Two Water Tanks System. FS means flow sensor.

Fault name	Fault description	Fault amplitude	Unit
F_a	Actuator fault in the pump	$[-0.1, -0.07] \cup [0.07, 0.1]$	$m \cdot s^{-1}$
F_{h_1}	Fault in sensor 1 measuring the water level h_1 in the upper tank, $\mathbf{T}_1$	$[-2, -1] \cup [1, 2]$	m
F_{h_2}	Fault in sensor 2 measuring the water level h_2 in the lower tank, $\mathbf{T}_2$	$[-2, -1] \cup [1, 2]$	m
F_{f_1}	Fault in sensor 3 measuring the flow f_1 between $\mathbf{T}_1$ and $\mathbf{T}_2$	$[-2, -1] \cup [1, 2]$	$m^3 \cdot s^{-1}$
F_{f_2}	Fault in sensor 4 measuring the flow f_2 out of $\mathbf{T}_2$	$[-2, -1] \cup [1, 2]$	$m^3 \cdot s^{-1}$
F_{l_1}	Leakage between $\mathbf{T}_1$ and sensor 3	$[0.4, 0.5]$	n/a
F_{l_2}	Leakage between sensor 3 and $\mathbf{T}_2$	$[0.4, 0.5]$	n/a
F_{l_3}	Leakage between $\mathbf{T}_2$ and sensor 4	$[0.4, 0.5]$	n/a
F_{c_1}	Partial obstruction (clogging) in the pipe between $\mathbf{T}_1$ and $\mathbf{T}_2$	$[0.4, 0.5]$	n/a
F_{c_2}	Partial obstruction (clogging) after $\mathbf{T}_2$	$[0.4, 0.5]$	n/a

TABLE 3.1: Possible Faults in the Water Tanks

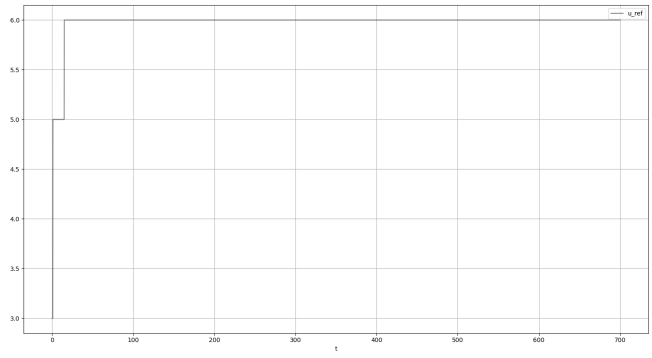


FIGURE 3.9: Graph of u_{ref} as a Function of Time (in $m^3 \cdot s^{-1}$)

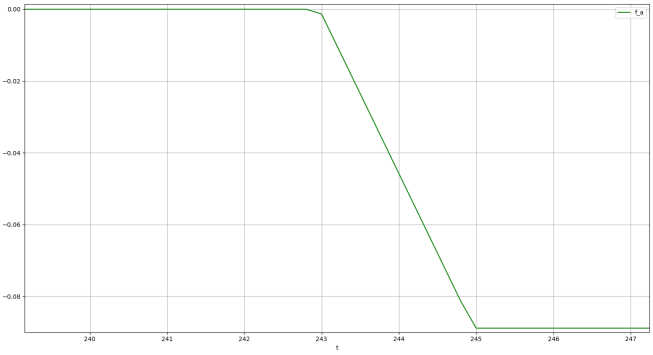


FIGURE 3.10: Example of a Fault Signal when the Fault Occurs

in Figure 3.10.

The water tanks system is ruled by the following equations:

$$e_1 : \dot{h}_1 = d_1 u_{ref} - d_2 x_2 + f_a \tag{3.3}$$
$$e_2 : \dot{h}_2 = d_3 x_1 (1 - f_{l_2}) - d_4 x_3 \tag{3.4}$$
$$e_3 : x_1 = (1 - f_{l_1}) x_2 \tag{3.5}$$
$$e_4 : x_2 = (1 - f_{c_1}) \sqrt{h_1} \tag{3.6}$$
$$e_5 : x_3 = (1 - f_{c_2}) \sqrt{h_2} \tag{3.7}$$
$$e_6 : \frac{dh_1}{dt} = \dot{h}_1 \tag{3.8}$$
$$e_7 : \frac{dh_2}{dt} = \dot{h}_2 \tag{3.9}$$
$$e_8 : y_1 = h_1 + f_{h_1} \tag{3.10}$$
$$e_9 : y_2 = h_2 + f_{h_2} \tag{3.11}$$
$$e_{10} : y_3 = d_5 x_1 + f_{f_1} \tag{3.12}$$
$$e_{11} : y_4 = d_6 (1 - f_{l_3}) x_3 + f_{f_2} \tag{3.13}$$
$$\tag{3.14}$$

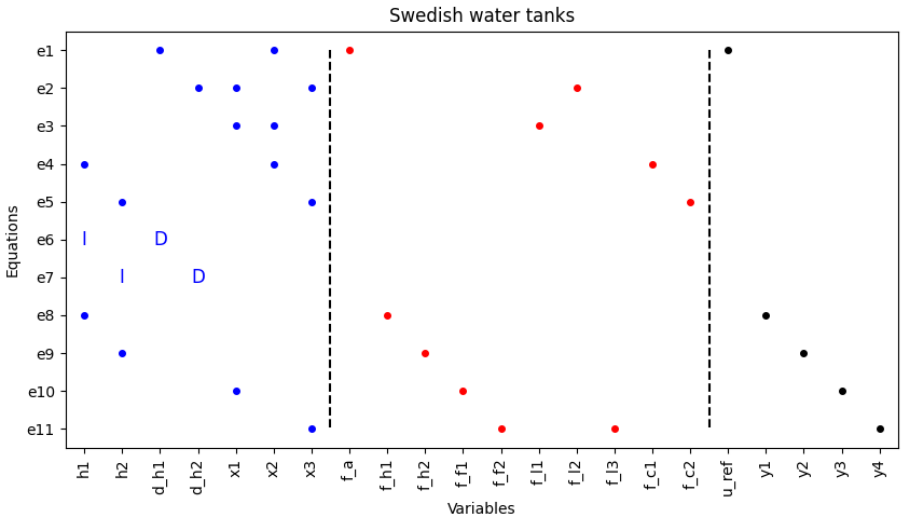


FIGURE 3.11: Structural Model of the Water Tanks. e_6 and e_7 are the differential constraints. I means that the variable is the primitive of the other, and D means that the variable is the derivative of the other.

3.4.2 Structural Analysis

3.4.2.1 Establish the Structural Model

The structural model of the water tanks is presented in Figure 3.11. It is established using the fault diagnosis toolbox (a Python tool described in Erik Frisk, Mattias Krysander, and Daniel Jung, 2017) from the aforementioned equations. It only uses structural relations and not physical equations of the system. In other words, it only uses knowledge of which variables are involved in each equation representing a component of the system.

3.4.2.2 Identify the MSO Sets and Compute the Fault Signature Matrix

From the structural model, we use the fault diagnosis toolbox to compute the fault signature matrix, presented in Figure 3.12 on the following page. It shows that there are seventeen MSO sets. From looking at the matrix, we can conclude that faults F_{f_2} and F_{l_3} are not isolable (see Definition 25 on page 29) because they have the same signature. This is also visible in the structural model, the faults appear in the same equation e_{11}.

3.4.2.3 Select a Subset with Maximal Isolability

Not all MSO sets are required to obtain maximal isolability of the faults. For instance, MSO sets MSO_2, MSO_8, MSO_9, MSO_{12}, MSO_{13} are enough, as shown in Figure 3.13 on page 41. They are enough to obtain maximal isolability because all faults have a different signature when considering only those MSO sets (except F_{f_2} and F_{l_3} that are not isolable from each other anyway).

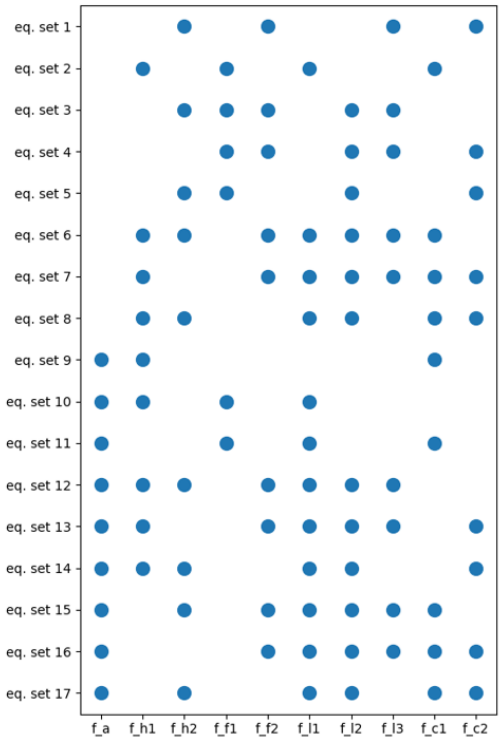


FIGURE 3.12: Fault Signature Matrix of the Water Tanks

	Observables	Fault Support
MSO_2	y_1, y_3	$f_{h_1}, f_{f_1}, f_{l_1}, f_{c_1}$
MSO_8	$y_1, y_2, \dot{y}_2$	$f_{h_1}, f_{h_2}, f_{l_1}, f_{l_2}, f_{c_1}, f_{c_2}$
MSO_9	$u_{ref}, y_1, \dot{y}_1$	f_a, f_{h_1}, f_{c_1}
MSO_{12}	$u_{ref}, y_1, y_2, y_4, \dot{y}_1, \dot{y}_2$	$f_a, f_{h_1}, f_{h_2}, f_{f_2}, f_{l_1}, f_{l_2}, f_{l_3}$
MSO_{13}	$u_{ref}, y_1, y_4, \dot{y}_1, \dot{y}_4$	$f_a, f_{h_1}, f_{f_2}, f_{l_1}, f_{l_2}, f_{l_3}, f_{c_2}$

TABLE 3.2: For each selected MSO set, list of involved variables and corresponding fault support.

There exists algorithms designed to automatically obtain a minimal subset of MSO sets that has maximal isolability properties (see Pérez-Zuñiga et al., 2018).

Only these five MSO sets are considered for the following steps of the method. From the structural model, it is possible to identify the observable variables that appear in the residual generators associated with each of these MSO sets. They are presented in Table 3.2. We can use this knowledge to train a machine learning algorithm to act as a residual generator. For instance, for MSO_2, we know that the residual generator involves only y_1 and y_3 and that data from faults $f_{h_1}, f_{f_1}, f_{l_1}, f_{c_1}$ should be predicted as not 0 whereas data from the other classes should be predicted as 0.

Note that, if we were to use the model equations, the ARR that could be obtained for MSO_2 would have been:

$$ARR_2 = y_3 - d_5\sqrt{y_1} \tag{3.15}$$

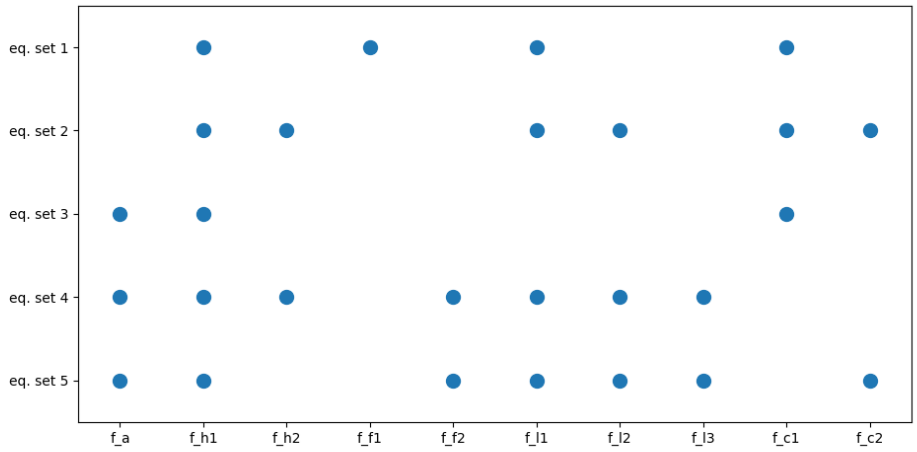


FIGURE 3.13: Fault Signature Matrix restrained to $MSO_2, MSO_8, MSO_9, MSO_{12}, MSO_{13}$.

3.4.3 Training for each MSO Set

3.4.3.1 Dataset Preprocessing

A dataset of 110 simulations is generated, 10 simulations per fault type (and 10 faultless simulations). The sampling frequency is 5Hz. No noise is injected in the dataset because derivatives of noisy data are generally mediocre, which is an orthogonal problem to our purpose, i.e. to show that the proposed diagnosis SA-ML architecture is interesting (see Section 5.2.1 on page 78). The dataset contains measurements of four observable variables, y_1, y_2, y_3, y_4, and a label l for the fault type (or nominal). First order derivatives of y_1, y_2, y_4[2] are computed using the `gradient` method of the NumPy package, and added to the dataset. Since u_{ref} is the input of the system, it is known and is also put in the dataset.

In the end, the dataset is split between a training and a testing dataset. The training dataset contains 308088 samples of 8 variables $(u_{ref}, y_1, y_2, y_3, y_4, \dot{y}_1, \dot{y}_2, \dot{y}_4)$ and the corresponding label l. The testing dataset contains 77022 samples.

For each MSO set, a specific dataset is used. Minor preprocessing is performed to have different labels depending on which MSO set is considered. Each MSO set has its own labeling:

$$l_i = \begin{cases} 0 \text{ if } l \notin FS_i \\ 1 \text{ if } l \in FS_i \end{cases} \tag{3.16}$$

with l_i the label for MSO_i with fault support FS_i. For each MSO set, this becomes a binary classification problem. Also, for each MSO set, only observable variables that are involved in the associated MSO are kept in the dataset.

3.4.3.2 Training

For each MSO set, five algorithms are trained for binary classification. The goal is to separate samples from the classes in the fault support of the ARR corresponding to

[2] $\dot{y}_3$ is not computed because structural analysis indicates that it does not appear in any of the residual generators selected.

Algorithm	MSO_2	MSO_8	MSO_9	MSO_{12}	MSO_{13}
sklDT	99.98	99.98	99.99	99.98	99.98
sklRF	99.99	99.98	99.99	99.98	99.99
sklLR	85.53	69.69	88.51	78.75	79.99
sklNB	73.47	71.11	22.29	63.63	58.97
sklKNN	99.98	99.83	99.96	99.95	99.97

TABLE 3.3: Accuracy of the Machine Learning Algorithms Simulating Residuals on a Testing Set (in %)

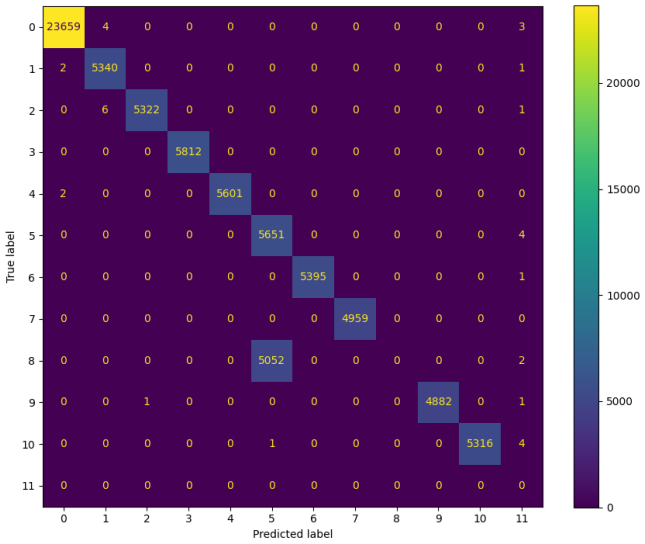


FIGURE 3.14: Confusion Matrix of the Hybrid AI Diagnosis Method

the MSO set from samples that are not labeled by classes in the fault support.

3.4.3.3 Best Algorithm Selection

The results from the training are presented in Table 3.3.

Clearly, decision tree, K-nearest neighbor and random forest classifiers are the best performing algorithms. Random Forest slightly outperforms the other two for every MSO set. Thus, each residual generator is simulated using a Random Forest binary classifier. A novel aspect of this proposed method over the existing ones is that the selected algorithm does not have to be the same for every MSO set. It is the case here because random forest outperforms the other algorithms on every MSO set. Also, for this study case, only five algorithms have been tested and perhaps other algorithms could have better performances than Random Forest.

3.4.4 Final Results

Using binary classifiers in conjunction with the signature matrix to predict the class of samples from the test set gives an accuracy of $93,40\%$ and the confusion matrix shown in Figure 3.14. Class 11 corresponds to samples that were predicted in an unknown class. This happens when there is no signature in the signature matrix that corresponds to the output of the machine learning classifiers.

Note that the non isolability of F_{l_3} and F_{f_2} impacts the accuracy. If those classes are considered as one class, the accuracy becomes 99.96%. The performances are really good for this method on this system. Furthermore, it only requires data and the structural model of the system. The structural model is a demanding requirement because it is hard to obtain, but this prerequisite is weaker than requiring the full analytical model of the system. However, the outcome of this method is simply a diagnosis, without any explanation as to why a fault occurred.

3.5 Conclusions

The study of existing hybrid AI diagnosis methods show that algorithms exist in this field but never use model-based methods to extract an explanation of the diagnosis. Also, most of the time, they still require some deep knowledge about the system (e.g. the structural model). Often, one step of a model-based algorithm is replaced with machine learning. No method tries to intrinsically take advantage of model-based concepts in a data-driven algorithm, which is what we explore in the next chapter.

We proposed a new hybrid AI diagnosis method, named SA-ML, heavily inspired by D. Jung, 2020. It relies on structural analysis but the residual generator computing part is replaced with machine learning algorithms. SA-ML is a variant allowing any kind of machine learning algorithm to replace the recurrent neural network and different algorithms to be selected for different faults. The method has been tested on the water tanks use case. The performance is very good but the diagnosis lacks explanation about the fault or knowledge learned about the system. However, decision trees show promising results by being one of the best algorithms in the context of the method while being partly explanatory.

Chapter 4

DT4X: Diagnosis Tree Enhanced with Meta-Knowledge

After exploring various options and experimenting with different machine learning techniques, a technology emerged as being able to synergistically combine data-based diagnosis methods with model-based diagnosis methods: symbolic classification. It allowed the design of a novel explainable diagnosis method called DT4X (Diagnosis Tree for eXplainability). DT4X leverages decision trees where decisions are informed by diagnosis meta knowledge, specifically focusing on the properties of diagnosis indicators. This knowledge is used at each node to articulate a symbolic classification problem, outputting discriminating functions. The outcome is a multivariate decision tree that produces a compact model for diagnosis. The use of decision trees increases the explainability of the outcome, all the more so as one discovers the explicit formal expressions of diagnosis indicators, structured in the form of analytical redundancy relations.

This chapter aims at explaining how DT4X works and the reasons behind its design. First, some background on genetic algorithms, symbolic classification and decision trees is required to understand the methods involved in DT4X.

4.1 Background

DT4X relies on symbolic classification. Symbolic classification itself relies on genetic algorithms.

4.1.1 Genetic Algorithms

Genetic algorithms come from the research of John Holland in 1960 but were popularized in the nineties (Holland, 1992). They are a kind of evolutionary algorithm and are inspired by the process of natural selection. They have been used to solve some very complex problems such as the Traveling Salesman Problem (Beardwood, Halton, and Hammersley, 1959) or the Knapsack Problem. They are often used to find an approximate solution to NP-hard problems.

Genetic algorithms are used to find approximate solutions to optimization and search problems. Let us define the main actors of a genetic algorithm.

Definition 29 (Candidate Solution)**.** *A possible solution to the problem at hand. Can also be called individual, creature, organism.*

Definition 30 (Population)**.** *The set of candidate solutions.*

Definition 31 (Fitness Function)**.** *A function that takes as input a candidate solution and outputs a score that represents how well it performs on the problem at hand. In other words, it evaluates how good an individual is in terms of solving the problem.*

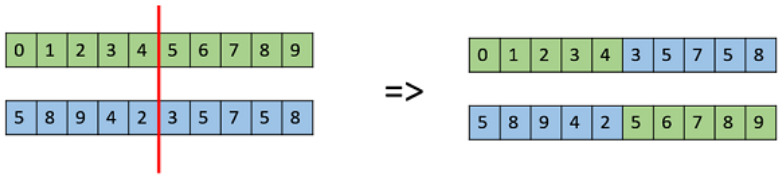


FIGURE 4.1: Example of a Crossover

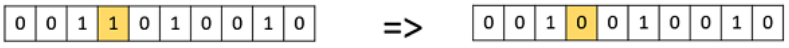


FIGURE 4.2: Example of a Mutation

The genetic algorithm iterates through generations (as in iterations or cycles of the algorithm's evolutionary process). Between each generation, the composition of the population changes. Individuals with the best scores at a given generation are selected and then used in a reproduction process to give the population of the next generation.

Definition 32 (Reproduction)**.** *The process of creating a new candidate solution. Selected individuals undergo genetic operations such as crossover and mutation to produce offspring. Crossover involves combining the genetic information of two parents to create new individuals, while mutation introduces small random changes in the genetic material.*

Figure 4.1 illustrates a crossover, while Figure 4.2 illustrates a mutation.

Using this process, the fitness of the population progressively improves over the generations. The algorithm terminates when a stopping criterion is met, such as reaching a maximum number of generations or when the score of an individual passes a given threshold. Of course, many variants of genetic algorithms exist, with different reproduction operators, stopping criteria, initialization and selection processes. Examples of use cases include search and optimization (Deb, 1998), machine learning (Grefenstette, 1993), robotics (Davidor, 1991), bioinformatics (Manning, Sleator, and Walsh, 2013) or even solving sudoku puzzles (Gerges, Zouein, and Azar, 2018).

4.1.2 Symbolic Classification

Symbolic classification is a technique in machine learning and symbolic artificial intelligence where symbolic expressions or rules are used to represent relationships between input variables and output labels. Unlike traditional statistical or machine learning approaches that rely on predefined mathematical models, symbolic classification aims to discover interpretable symbolic expressions directly from the data.

Figure 4.3 illustrates the training process of symbolic classification. It consists in estimating a function $f : \mathbb{R}^n \to \mathbb{R}$ knowing samples (x, l) with $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ and $l \in C = \{0, 1\}$. C represents a set of classes of cardinality two (classes do not need to be 0 and 1 but they can be mapped to $\{0, 1\}$ so for clarity purposes we only

FIGURE 4.3: Symbolic Classifier: during **training**, the green objects are known and the red ones are unknown.

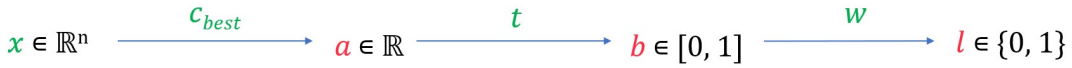


FIGURE 4.4: Symbolic Classifier: during **testing**, the green objects are known and the red ones are unknown

consider $C = \{0, 1\}$). f is the actual function, the one that symbolic classification is trying to find the expression of, or at least get as close as possible to. Let us call c_{best} the function found by symbolic classification that tries to be as similar to f as possible. c_{best} is then used together with a *classification function*[1] $t : \mathbb{R} \to [0, 1]$ to predict the class of a sample x. The output of t is in the real interval $[0, 1]$ but a threshold function $w : [0, 1] \to \{0, 1\}$ can be applied to obtain the class in $\{0, 1\}$. In other words, $w \circ t \circ f(x)$ gives the class of x. Figure 4.4 shows how prediction of the class of a new sample x is performed once c_{best} has been found.

Definition 33 (Symbolic Classifier)**.** *We call symbolic classifier the composition of* c_{best}*,* t *and* w*.* $w \circ t \circ c_{best} : \mathbb{R}^n \to \{0, 1\}$*.*

The classification function t can take many shapes. Its default value usually is a sigmoid function (Han and Moraga, 1995). In the context of this manuscript, the classification function is customized to fit our needs (see Section 4.2.2.2 on page 57).

The estimation of f is done without assuming its structure; an analytical relation is therefore just discovered. There are multiple ways to carry out this estimation. Symbolic classification algorithms typically explore a vast space of possible expressions to find the most accurate model. Evolutionary algorithms, such as genetic algorithms, are commonly used for this purpose due to their ability to efficiently search complex solution spaces. To the best of our knowledge, symbolic classification is only implemented in the python package `gplearn` (Stevens, 2016) which uses a genetic algorithm to estimate f. `gplearn` is based on (Poli, Langdon, and McPhee, 2008). For instance, the paper (S. Liu et al., 2022) (despite being completely unrelated to our topic) uses symbolic classification through `gplearn`. Nevertheless, it is clear that some other methods could be used to estimate f in the context of symbolic classification and it is discussed in Section 6.1.1 on page 91.

Thus, in our case, symbolic classification relies on a genetic algorithm that takes as inputs a set of samples $\mathcal{D} = \{(x, l)\}$ called the dataset and a set of operators O (e.g. $+, *, -, /, \sqrt{}, ||, \log$, etc.). The genetic algorithm searches for the best expression c_{best} combining variables $(x_1, \ldots, x_n)$ and operators so that $\sum_{x \in \mathcal{D}} |w \circ t \circ c_{best}(x) - w \circ t \circ f(x)|$ is minimal. It generates candidate solutions in the form of expressions $c : \mathbb{R}^n \to \mathbb{R}$. These expressions are represented as expression trees (Preiss, 1998). Figure 4.5 on the next page shows an example of such an expression tree. If a node is an operator, it has as many edges coming out of it as its arity (number of operands). Each child node is one of its operands. If a node is a variable or a constant, it is a leaf. The candidate expressions are evaluated using a fitness function. The fitness function can vary depending on the application of symbolic classification, but is most often the log loss function (Bishop, 2006), given by:

$$Fitness(c) = -\frac{1}{|\mathcal{D}|} \sum_{(x,l) \in \mathcal{D}} [l \ln(t(c(x))) + (1 - l) \ln(1 - t(c(x)))] \tag{4.1}$$

[1] The notation t comes from the original name of this function: the *transformer*. However, since we are dealing with Artificial Intelligence related topics, it could be confusing to name this function a *transformer*. Also, *classification function* really fits the role of the function which is to turn a regression problem into a classification problem by fitting classes rather than a variable.

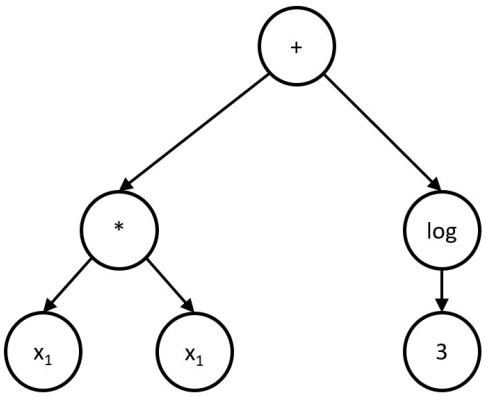


FIGURE 4.5: Expression Tree of $x_1^2 + log(3)$

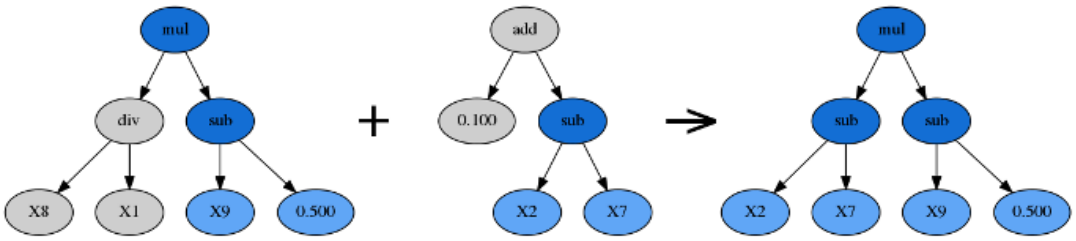


FIGURE 4.6: Crossover Between Two Candidate Expressions

Following the principles described in 4.1.1 on page 45, an initial population of candidate expressions is randomly generated. Then, the individuals are evaluated using the fitness function. A new generation of individuals is generated by reproducing the previous ones. Candidates with a high fitness are more likely to be used in the reproduction process. Reproduction can consist of various operations. The ones used in `gplearn` are:

- crossover, consisting in merging parts of two expression trees together (Figure 4.6);
- hoist mutation, consisting in replacing the subtree of a candidate expression by a new node (Figure 4.7 on the facing page);
- subtree mutation, consisting in replacing the subtree of a candidate expression by a new subtree (Figure 4.8 on the next page);
- point mutation, consisting in replacing a node of a candidate expression by a different node, of the same nature or not (Figure 4.9 on the facing page). It can incur changes in the subtree that starts from this node if the new node has a different arity.

This process is repeated until a stopping criterion is reached. It can be a stagnating fitness or a fitness threshold being reached by a candidate expression.

4.1.3 Decision Trees

A decision tree is a popular supervised learning algorithm used for both classification and regression tasks in machine learning.

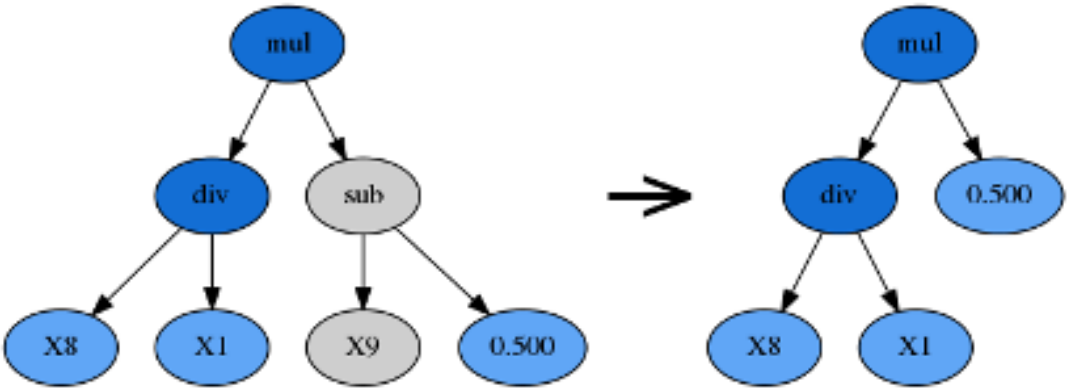


FIGURE 4.7: Hoist Mutation of a Candidate Expression

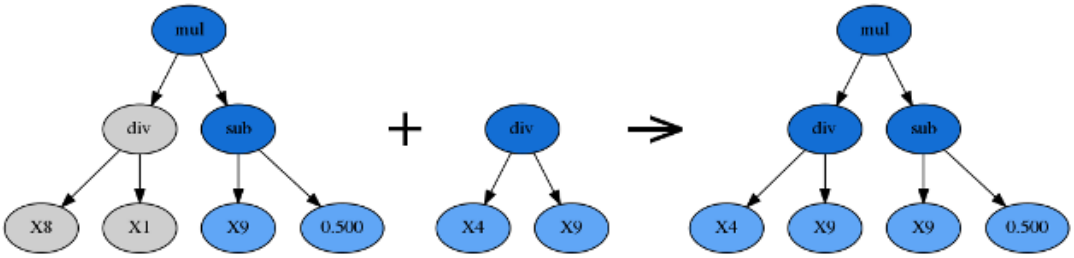


FIGURE 4.8: Subtree Mutation of a Candidate Expression

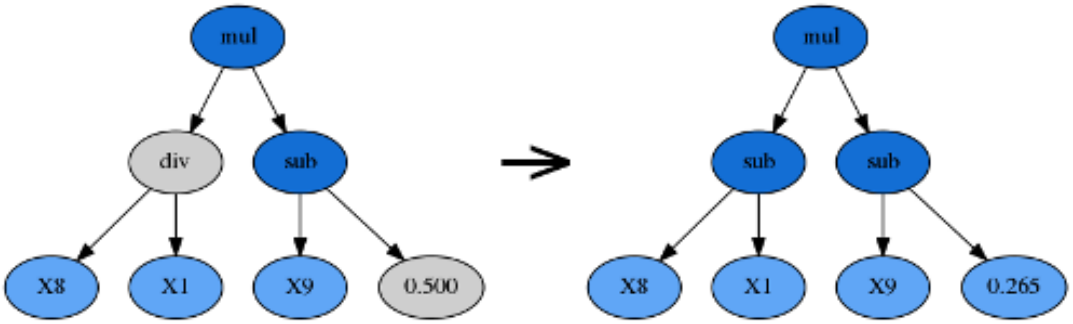


FIGURE 4.9: Point Mutation of a Candidate Expression

Definition 34 (Decision Tree). *A decision tree $T(E, N)$ is a directed acyclic graph having at most one edge between every pair of nodes (Fürnkranz, 2010). E is the set of edges and N is the set of nodes. T must have exactly one root node n_0. All other nodes have exactly one incoming edge.*

Definition 35 (Root Node). *A root node n_0 is the only node of a decision tree with no incoming edge.*

Definition 36 (Leaf). *Nodes without outgoing edges are called leaves.*

Definition 37 (Binary Decision Tree). *Specific type of decision tree where each node that is not a leaf has exactly two outgoing edges.*

Definition 38 (Children of a Node). *The children of a node are the nodes reachable from it by following the two outgoing directed edges. Leaves do not have children.*

Definition 39 (Path to a Node). *The path to a node is the ordered list of edges followed to reach this node from n_0.*

Theorem 4.1.1 (Unicity of the path). *The path to a node is unique. (Fürnkranz, 2010)*

Corollary 4.1.1.1 (Unicity of the path to a leaf). *The path to a leaf is unique.*

4.1.3.1 Training a Decision Tree

Training a decision tree involves building the structure of the tree based on a dataset with known input-output pairs. It starts from the root node n_0 in which the training dataset is stored.

Definition 40 (Purity of a Node). *A node n_i is said pure when the dataset associated to it contains samples from only one class C_i.*

During the training phase, a process is repeated in each node of the tree, starting from n_0. Let us consider a node n_i that contains the dataset $\mathcal{D}_i$. If n_i is pure and contains samples from the class C_i, it is declared a leaf and labeled with that class C_i. If n_i is not pure, a discriminating criterion $\mathbf{d}_i$ is computed. The most common ways to determine if a discriminating criterion is good are *gini* and *entropy* (Priyam et al., 2013). They are a measure of the purity of a node. Thus, they can measure the purity of the child nodes that would be created according to a given criterion. For instance, the gini impurity measure of a node n_i containing a dataset $\mathcal{D}_i$ made of samples from classes $C_1, \ldots, C_k$ is:

$$gini(n_i) = 1 - \sum_{C \in \{C_1, \ldots, C_k\}} \left(\frac{|C|}{|\mathcal{D}_i|}\right)^2 \tag{4.2}$$

If $k = 1$, the node n_i contains only one class and $|C_1| = |\mathcal{D}_i|$, meaning that $gini(n_i) = 0$, which corresponds to the best possible purity. Entropy follows the same principles. In this manuscript, only gini is used. These metrics are used to determine which criterion to use to split the dataset in order to maximize purity of the children. Once $\mathbf{d}_i$ is chosen, all samples in n_i are evaluated on $\mathbf{d}_i$, leading to the creation of two new nodes: one for the samples that satisfy the criterion ($\{x \in \mathcal{D}_i, \mathbf{d}(x)\}$), and one for those that do not ($\{x \in \mathcal{D}_i, \neg\mathbf{d}(x)\}$). This process is repeated for newly created

nodes until there are no impure nodes left[2]. As a consequence, when the decision tree is fully trained, each leaf is assigned a class.

The condition $\mathbf{d}_i$ of a non-leaf node n_i, determined during training, can be presented this way: n_i is associated with $k_i \in \mathbb{N}$ features. n_i is also associated with a subset of $\mathbb{R}^{k_i}$. The condition $\mathbf{d}_i$ is verified when the actual values of the k_i features belong to the associated subset and not verified otherwise.

4.1.3.2 Using a Tree

Once the decision tree has been trained, it can be used to predict the class of a sample x. Classification of a sample x is performed by following the path of conditions that correspond to its feature values until reaching a leaf node. The predicted class is the one associated with this leaf node.

4.1.3.3 Multivariate vs Univariate

Decision trees can be split into two distinct categories: univariate and multivariate.

Definition 41 (Univariate Decision Trees)**.** *Univariate decision trees are decision trees whose criteria are only based on one feature at a time. This means that $k_i = 1$ for all non-leaf nodes.*

Definition 42 (Multivariate Decision Trees)**.** *Multivariate decision trees are decision trees whose criteria can be based on any number of features (and it it not fixed across the tree). For all non-leaf nodes, $k_i \in [1, n_{\mathbf{x}}]$ (Brodley and Utgoff, 1995).*

With these definitions, univariate decision trees are a particular case of multivariate decision trees. In the literature, the most common type of decision trees are univariate decision trees. The recent survey of Costa and Pedreira, 2023 calls "traditional trees" the univariate decision trees from Priyam et al., 2013 and says that it "remains one of the most popular algorithms in the field, and also serves as the base for several other contributions." For diagnosis purposes, multivariate decision trees are of utmost importance because faults can be detected by comparing input and output observable variables and not only one variable at a time.

4.2 DT4X

DT4X is the algorithm designed as a solution to our initial problem (see Section 1.1 on page 1). Our ambition is to intrinsically combine model-based and data-driven diagnosis methods. DT4X is a data-driven diagnosis methods in the sense that it uses data to make predictions on the state of the system. It is also a model-based method since the data is used to find equations of the system model (namely the data-driven equivalent of analytical redundancy relations presented in Section 3.1 on page 29) and then use them to compute a diagnosis. To be more precise, it is not a model-based method *per se*, but rather, it uses meta-knowledge from model-based methods in order to find analytical model-based relations using data.

DT4X stands for Diagnosis Tree for eXplainability. It also stands for Diagnosis Tree with 4 main properties (4X): multivariate analysis, explicable decision-making, incorporation of meta-knowledge and use of symbolic classification.

This section goal is to explain what is DT4X and how it is implemented.

[2] In practice, most decision tree algorithms allow for a percentage of the data in the node to not be of the same class. Otherwise, the decision tree can quickly become huge and prone to over-fitting.

4.2.1 Principle

DT4X aims at training a binary decision tree to diagnose a system. The inputs for DT4X are a training dataset $\mathcal{D}$, a list of operators and the hyper-parameter (see Section 4.2.2.3 on page 58) values.

$\mathcal{D}$ must contain data from the nominal class and from all the faulty classes. Indeed, the output tree is only able to predict faults used to train it. It is preferable to have more nominal data than data from the other classes. It is often the case since it is easier to acquire nominal data for pretty much any system. A sample x of $\mathcal{D}$ contains values of observable variables. This x is also called a *feature vector* and the variables inside the *features* of a sample. This term is often used when dealing with machine learning in general.

The operators are a set of functions specified by the user. The usual operators are: $+, *, -, /, sign, abs, \sqrt{}, cos, sin$, ... By default, the operators are $+, -, *, /$. They should be chosen according to system behavior knowledge. For instance, if a component is known to square its input, it makes sense to include the square operator and its inverse, the square root operator. In this way, expert knowledge is injected into the algorithm. It can come from some system model knowledge (which links back to the idea of combining model-based and data-driven diagnosis).

In order to inject meta-knowledge from model-based diagnosis into DT4X, we use the concept of diagnosis indicator. It coincides with the definition of residual generator of Section 3.1 on page 29.

Definition 43 (Diagnosis Indicator)**.** *A diagnosis indicator (or fault indicator) is a relation that maps quantities that have been observed, or those deducible from observations, into a scalar value, providing an indication of a fault. A relation of the form* $\mathbf{d}(x', \dot{x}', \ddot{x}', ...) = r$*, with input* x'*, a subvector of* x *the observable variables, and output* r*, a scalar named* residual*, is a* diagnosis indicator $\mathbf{d}$ *if for each* x *measured on the system in nominal conditions, the relation is fulfilled, i.e.* $r = 0$ *(or tends to 0 when time tends to* ∞ *in the case of a dynamic system).*

Definition 44 (Evaluate)**.** *To evaluate a diagnosis indicator* $\mathbf{d}$ *on a sample* x *means to compute the image of* x *by* $\mathbf{d}$*.*

A diagnosis indicator evaluated on a nominal sample should be zero, considering an ideal, non-noisy, environment, and non-zero on faulty samples. The main idea of DT4X is to use this knowledge to find appropriate multivariate relations that are used as branching conditions to take a decision in each node of the decision tree. To do so, symbolic classification is used with a modified classification function (see Section 4.2.2.2 on page 57). This constraints the output function to be a diagnosis indicator in the sense that it has the same properties and discrimination capabilities.

In the decision tree resulting of DT4X, each node $n_i \in N$ that is not a leaf therefore contains a binary diagnosis indicator $\mathbf{d}_{n_i} : \mathbb{R}^n \to \{0, 1\}$. Each diagnosis indicator $\mathbf{d}_{n_i}$ is used to partition the data into two disjoint subsets, depending on whether $\mathbf{d}_{n_i}(x) = 0$ or $\mathbf{d}_{n_i}(x) = 1$ for all x belonging to n_i. The two subsets are then sent to a different child node. Each leaf of the resulting tree has a label that is the class predicted for the data reaching this leaf. See Figure 4.10 on the facing page for an example. In each node, except the root node, the first line indicates the node number. For non-leaf nodes, the next line shows the found diagnosis indicator. Then, the following line shows the classes used to train symbolic classification (to find the diagnosis indicator). Leaves contain their diagnosis, which means the class label given to samples that reach this leaf. All nodes except the root node contain their own gini purity value. For the root node, it is always bad because all the classes are

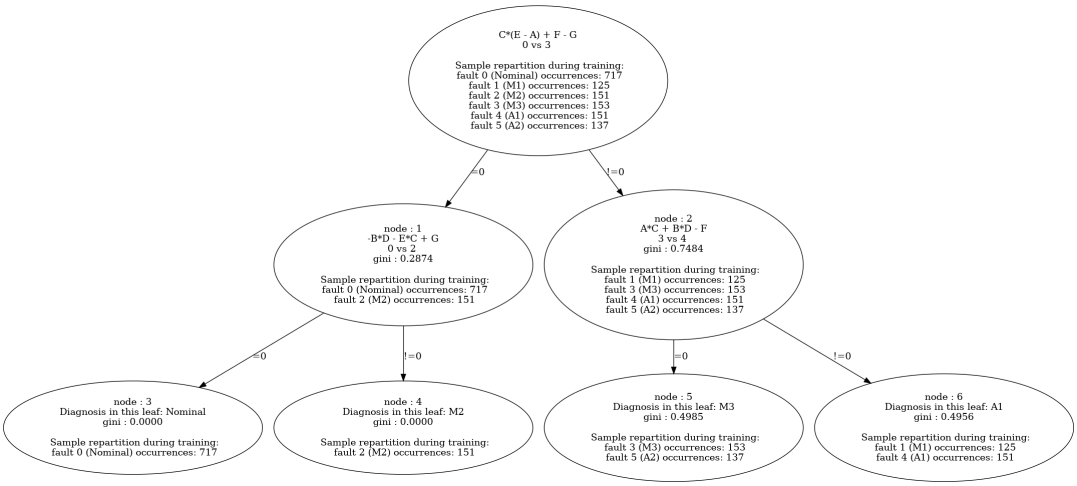


FIGURE 4.10: Example of a Decision Tree Produced by DT4X

present, hence why it makes no sense to display it. Finally, for each node, the sample distribution is given.

4.2.2 DT4X Algorithm

The algorithm uses concepts that need to be defined:

Definition 45 (Pure with label l)**.** *A node n_i is said to be pure with label l if at least X_p% of the samples belonging to n_i are of class l.*

This corresponds to the definition of purity given in Definition 40 on page 50, but extended to allow leaves to mainly contain one class while still having some samples from other classes. The value X_p is a hyper-parameter of DT4X.

Definition 46 (Relevancy in n_i)**.** *A class is said to be relevant in n_i if the amount of samples of this class present in n_i constitutes at least X_r% of the whole dataset.*

The value X_r is a hyper-parameter of DT4X.

Algorithm 1 on the following page gives the pseudo-code of DT4X. The arrow symbol with a plus ($\leftarrow +$) means that the value is appended to the variable.Next sections explain the algorithm in details. All the keywords used are explained in Section 4.2.2.1.

4.2.2.1 Detailed Explanation

During the training phase, a node $n_i \in N$ contains a subset of samples $\mathcal{D}_{n_i} \subset \mathcal{D}$. Each sample (x, l) in n_i verifies the conditions defined on the edges leading to n_i from the root node. At the beginning of DT4X (line 1), the root node n_0 contains the entire set $\mathcal{D}$ (see Figure 4.11 on page 55).

DT4X builds the tree starting from the root node and then going through every single node in their order of creation. The algorithm stops when no nodes are left to deal with (line 2).

When reaching a node n_i, DT4X first checks whether n_i is `pure with label` l (line 4) (see Definition 45). If it is the case, n_i is designated as a leaf and the label l is associated with it (lines 5 and 6).

Otherwise, the goal is to find a new diagnosis indicator $\mathbf{d}_{n_i}$ that splits the data belonging to $\mathcal{D}_{n_i}$ (line 8 to 15). Thus, let us `generate` a set of possible pairs of classes ($pairsToTry$) to split using a symbolic classifier (line 8).

Algorithm 1 DT4X pseudo-code

Inputs: $\mathcal{D}$, O, Hyper-Parameters, Untrained Decision Tree (n_0)
Output: Trained Decision Tree with Diagnosis Indicators

$currentNodes \leftarrow n_0$
while $currentNodes$ is not empty **do**
  for all $node \in currentNodes$ **do**
    if $node$ is `pure with label` l **then**
      $node$ is leaf
      $node \leftarrow l$
    else
      $pairsToTry \leftarrow$ `generate` $pairs$
      $pair \leftarrow$ first element of $pairsToTry$
      while not `check` c_{best} and $pairsToTry$ not empty **do**
        `balance` $pair$
        $c_{best} \leftarrow$ `Symbolic Classification` on $pair$
        $pair \leftarrow$ next element of $pairsToTry$
      end while
      if c_{best} **then**
        $lNode, rNode \leftarrow$ `split` according to c_{best}
        $futureNodes \leftarrow +lNode, rNode$
      else
        $node$ is leaf
        $node \leftarrow$ `majority` label l
      end if
    end if
  end for
  $currentNodes \leftarrow futureNodes$
end while

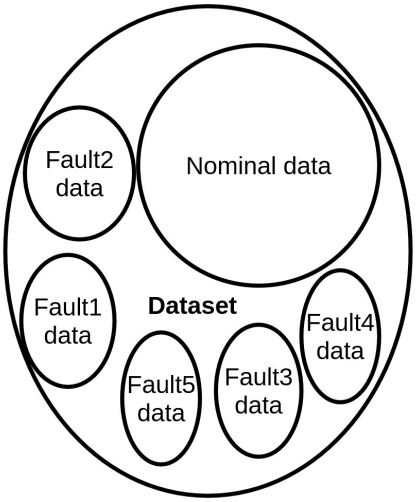


FIGURE 4.11: The Root Node n_0

Two cases are distinguished for the pair generation.

If the nominal class is relevant (see Definition 46 on page 53) in the node n_i, the set of pairs to try ($pairsToTry$) is built by having the nominal data as the first class, and any of the faulty classes relevant in n_i as the second class. This means pairs of the shape $(nominal, f_k)$.

If the nominal class is not relevant in n_i, the first step is finding the set of faulty classes that are relevant in n_i. Then, all permutations of pairs of these classes are generated. For p faulty classes, this results in $p * (p-1)$ pairs. It also means that if the pair (f_i, f_k) is present, the pair (f_k, f_i) is also present. This is important because the following step of the algorithm modifies the first class of the pair. During the balancing process (see further), half of the data of the first class is made of samples of the class itself, and the other half is made of nominal data randomly selected from the initial dataset. This ensures that symbolic classification finds an expression that is worth 0 for both nominal samples and samples belonging to the first class of the pair. This expression is also different from 0 for samples of the second class of the pair. In other words, the expression is triggered by samples of the second class but not by samples of the first or by nominal samples.

Once the set of pairs ($pairsToTry$) is generated, the algorithm loops over these pairs until either it runs out of pairs to try or until a diagnosis indicator $\mathbf{d}_{n_i}$ is found that discriminates the classes from the pair correctly (lines 10 to 11). Figure 4.12 on the next page shows the first pair selected.

When a pair is selected from $pairsToTry$, step one is to `balance` samples in the two classes (line 12). This preprocessing checks which class in the pair has fewer samples and randomly selects the same number of samples from the class that has the most samples. This is done to ensure symbolic classification is performed with balanced classes. See Figure 4.13 on the following page.

Then, the symbolic classification algorithm is performed on the pair as described in Section 4.1.2 on page 46 (line 13, see Figure 4.14 on page 57). The classification function is described in Section 4.2.2.2 on page 57. Symbolic Classification always returns a candidate expression, named c_{best}, that is the best expression found with respect to the fitness (see Section 4.1.2 on page 46) (line 13). However, while this expression might be the best found, it might not necessarily qualify as a good diagnosis indicator, either because the best possible expression has not been found or because

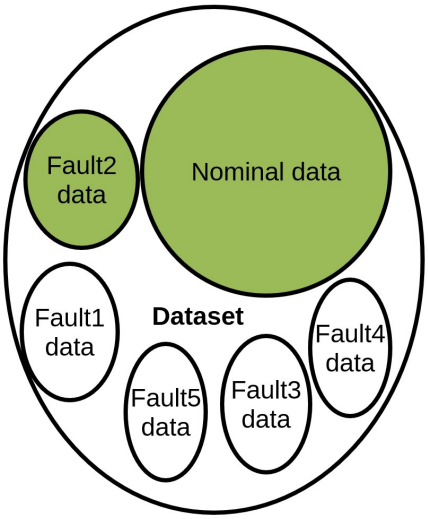


FIGURE 4.12: The First Pair Selected

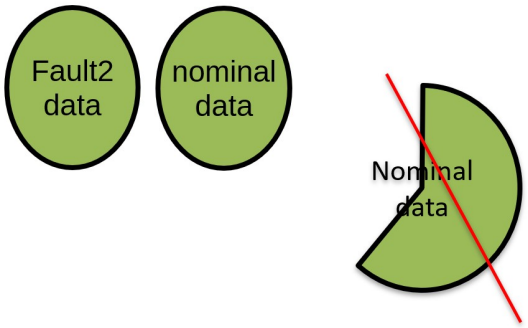


FIGURE 4.13: Balancing the Pair

the pair of classes used to train the symbolic classification are samples from non isolable fault cases. Thus, it is important to `check` (line 10) if the found expression c_{best} is a valid diagnosis indicator. This is done by taking two consecutive tests.

T1 checks that the nominal data from the whole dataset is predicted as 0 by c_{best}. If at least $X_{T_1}\%$ of the nominal data is predicted as 0 then the test is passed successfully, X_{T_1} being a hyper-parameter of DT4X.

T2 checks that c_{best} correctly predicts $X_{T_2}\%$ of the data used to find it through symbolic classification. This percentage does not include data discarded during the balancing process. This test ensures that c_{best} is classifying correctly. X_{T_2} is a hyper-parameter of DT4X.

If either **T1** or **T2** is false, then c_{best} is not considered as a valid diagnosis indicator and the loop over the pairs continues.

Once either a valid diagnosis indicator $\mathbf{d}_{n_i}$ has been found or all pairs have been tested, the while loop is exited. If a diagnosis indicator $\mathbf{d}_{n_i}$ was found, corresponding to c_{best} (line 16), the data in node n_i is `split` according to $\mathbf{d}_{n_i}$ (see example in Figure 4.15 on the next page). The algorithm evaluates $\mathbf{d}_{n_i}$ on the samples within $\mathcal{D}_{n_i}$. If the result is 0, the sample (x, l) is sent to the left child of the current node ($lNode$). If the result is different from 0, the sample is sent to the right child ($rNode$). In traditional decision trees (Section 4.1.3 on page 48), gini is used to evaluate how good a discriminating criterion is. Here, a discriminating criterion (a diagnosis indicator) is accepted right away without measuring how well it splits the

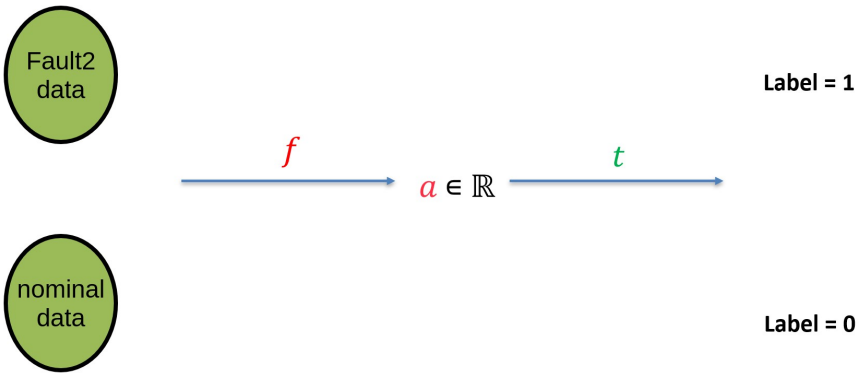


FIGURE 4.14: Symbolic Classification on the Pair: the nominal samples are labeled 0 and the faulty samples are labeled 1. The goal is to fit f while t is known.

child nodes. Instead, the tests **T1** and **T2** have been performed to make sure $\mathbf{d}_{n_i}$ is a diagnosis indicator, which means it splits some classes in a relevant way. However, some cases can occur such as with *Fault1 data* on Figure 4.15 where classes are not placed whole in one child node or the other.

If no diagnosis indicator was found (line 20), the class with the most samples in $\mathcal{D}_{n_i}$ has the `majority`, and the node is labeled with this class (line 22) and declared a leaf.

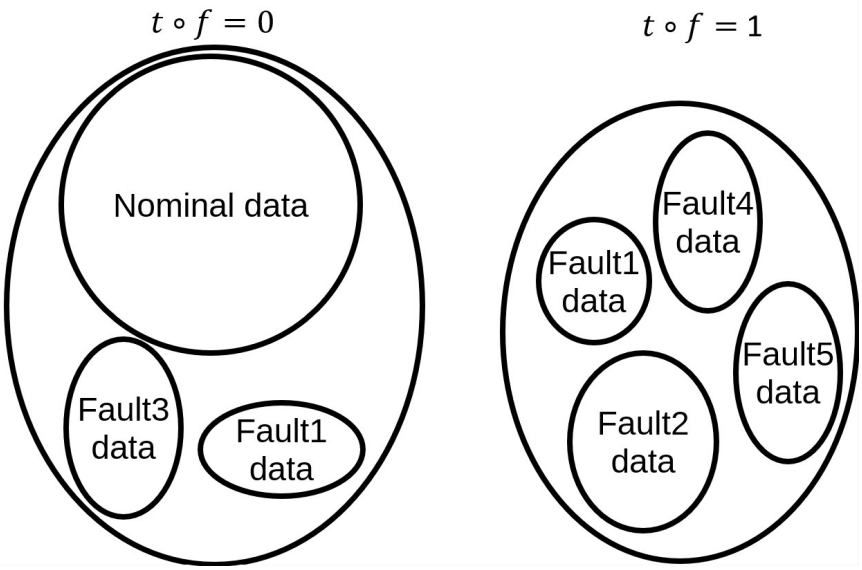


FIGURE 4.15: Splitting According to $\mathbf{d}_{n_i}$ (f in this case)

4.2.2.2 Classification Function

The classification function t for symbolic classification is customized for discovering diagnosis indicators:

$$\text{If } |a| < \epsilon, t(a) = 0, \text{ otherwise } t(a) = 1, \tag{4.3}$$

with $a \in \mathbb{R}$ and ϵ a DT4X hyper-parameter. See Figure 4.16 on the following page. If nominal data is inputted as class 0 and a faulty scenario data as class 1,

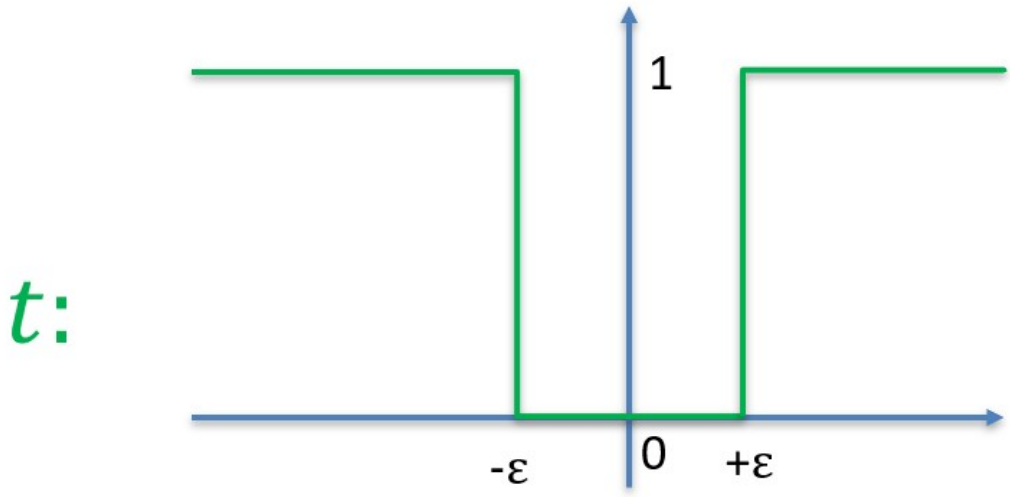


FIGURE 4.16: Classification Function used for Symbolic Classification in DT4X

and a function is discovered with high accuracy on this data, then this function is a diagnosis indicator, as it is characterized by being null for nominal cases, non-null for the class 1 faulty scenario, and involving only observable variables. Such indicator is sensitive to, at least, the fault used to find it.

In our case, there is no need for a threshold function w (such as defined in Section 4.1.2 on page 46) because t already gives values in $\{0, 1\}$.

4.2.2.3 Hyper-Parameters

A crucial parameter for DT4X is the list of operators that are used to estimate the diagnosis indicators using symbolic classification. They can be chosen according to prior knowledge about the system (which operators are included in the component equations, for instance). More operators, and more complex operators increases the time complexity of DT4X (see Section 4.2.2.6 on page 60) so they should be inputted with parsimony. The way symbolic classification is implemented in gplearn allows for any python function to be considered an operator. This opens infinite possibilities.

The hyper-parameters for DT4X are summarized in Table 4.1.

DT4X hyper-parameter	Default Value
purity threshold X_p	0.95
relevance threshold X_r	0.001
performance on nominal threshold X_{T_1}	0.95
indicator performance threshold X_{T_2}	0.90
Symbolic Classification	**Default Value**
ϵ	0.01
population size	5000
maximum number of generations	50
stagnation number	4
proportion of samples used	1
parsimony coefficient	0.02

TABLE 4.1: List of DT4X hyper-parameters and their default values

The DT4X hyper-parameters are described in Section 4.2.2.1 on page 53.

We now describe the symbolic classification hyper-parameters.

The ϵ parameter (Section 4.2.2.2 on page 57) has a powerful influence on the outcome of symbolic classification, so it should be modified according to the studied system. It should be scaled according to the order of magnitude of the data.

The *population size* parameter is the number of candidate solutions generated at each generation. A higher value reduces the number of generations before reaching a good solution but it extends training time.

The *maximum number of generations* is the number of generations beyond which symbolic classification stops, even if it has not found a solution. High value means more chances to find the right solution, but when no solution is to be found, it lengthens the time it takes to stop. However, the *maximum number of generations* should be high enough to never be reached. Indeed, there is another stopping condition called the *stagnation condition*. If the fitness value of the best candidate solution is the same for *stagnation number* generations in a row, the symbolic classification is stopped. This either occurs when the fitness reaches its minimum value or when the best candidate solution remains the same for a while (stagnates) without improving.

The *proportion of samples used* is the dataset fraction used to test each candidate solution. It provides a trade-off between computation time and accuracy. Higher accuracy means finding better diagnosis indicators, leading to faster predictions. Indeed, prediction time should have priority over training time. Thus, the whole dataset is used by default.

When computing fitness for a candidate solution, a penalty is added to its score. This penalty is the *parsimony coefficient* multiplied by the expression length of the candidate solution, favoring shorter solutions (because the fitness aims to be as small as possible).

4.2.2.4 Refitting

Once the decision tree is fully trained, it can sometimes be further improved. Indeed, DT4X relies on a genetic algorithm that, while very likely, does not guarantee convergence (Holland, 1992). Reusing symbolic classification on a pair that initially did not produce a diagnosis indicator might produce one with, for example, a different seed for the randomization of mutations. Thus, a `refit` function has been designed to automatically select leaf nodes with the lowest purity scores and retry symbolic classification with all the relevant classes in those nodes. It uses the *gini* metric described in Section 4.1.3 on page 48 to measure the purity of leaf nodes and selects the most impure leaves based on a threshold. Then, it reruns DT4X by initializing the variable *currentNodes* (see line 1 of Algorithm 1 on page 54) to the list of impure nodes to refit. Algorithm 2 on the following page gives the pseudo-code of `refit`. The `refit` function also allows adjustments to symbolic classification parameters, such as the mutation rate or the population size. `refit` is usually performed only once because after two tries, an impure node most likely contains classes that are non-isolable from each other.

4.2.2.5 Implementation Architecture

The UML graph of DT4X architecture is presented in Figure 4.17 on page 61. The algorithm is implemented in Python using pandas, numpy and sympy. The models are saved using pickle. The tree is displayed using graphviz.

Algorithm 2 Refit

Inputs: $\mathcal{D}$, O, Hyper-Parameters, Decision Tree Trained by DT4X, Threshold $\mathcal{T}$
Output: Trained Decision Tree with Diagnosis Indicators

```
refitNodes ← emptyList
for all node ∈ tree do
    if gini(node) ≥ T then
        refitNodes ← +node
    end if
end for
DT4X(currentNodes ← refitNodes)
```

4.2.2.6 Time Complexity

A question that might arise when considering a novel algorithm such as DT4X is "How long does it take to get a result ?" Time complexity is a very insightful indicator of that and in particular the worst-case time complexity gives an upper-bound of how long it takes to obtain the desired output of an algorithm. Multiple factors play a role in the overall time complexity of DT4X. The most time-consuming process is evaluating an expression on a sample of data. Here, we count the number of times this operation is performed. It is called $\mathcal{C}$. The factors involved are recapped in Table 4.2.

Factor	Variable Name
number of nodes in the tree	n_n
number of samples in the dataset	$n_{\mathcal{D}}$
average length of a candidate solution during symbolic classification	l_{avg}
maximum number of generations during symbolic classification	n_g
number of candidate solutions for each generation of symbolic classification	n_{cs}
number of classes in the dataset	n_C
number of input operators	n_O
number of input variables	n_x

TABLE 4.2: List of Variables that Impact Time Complexity

We consider the worst case scenario in which all nodes of the tree contain the whole dataset. Also, we consider that all the possible pairs of classes are tested in each node and that the last generation of symbolic classification is reached every time. Finally, we consider the amount of samples on which symbolic classification is trained to always be the whole dataset.

In every node, the same costly operations occur, namely:

1. for each pair:
   1a. running symbolic classification
   1b. testing the obtained expression
2. creating new nodes if applicable

Let us take a look at the cost of each operation in detail:

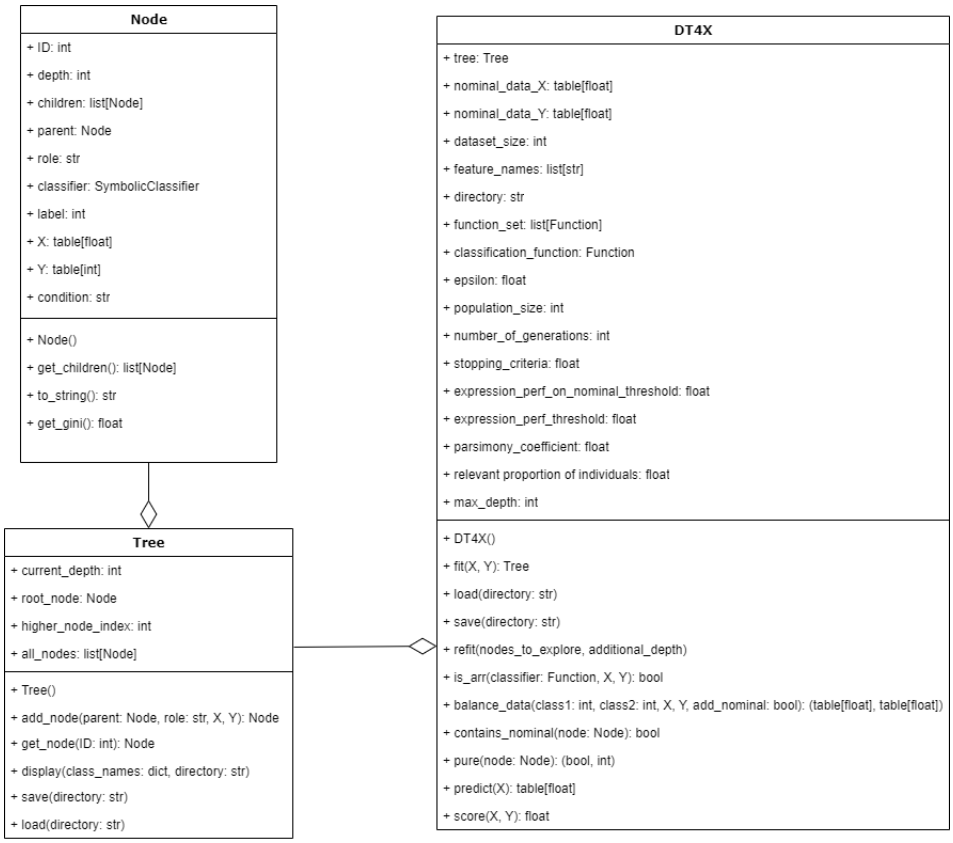


FIGURE 4.17: UML graph of DT4X

Operation 1a. During each generation of symbolic classification, each candidate solution is tested on each sample. At worst, this means $n_g n_{cs} n_{\mathcal{D}}$ expression evaluations.

Operation 1b. The obtained expression is tested on the training dataset of symbolic classification and on the nominal samples of the whole dataset. At worst, this means $2n_{\mathcal{D}}$ expression evaluations.

Operation 2. The newly found diagnosis indicator is evaluated on the node dataset. At worst, this means $n_{\mathcal{D}}$ expression evaluations.

Operations 1a and 1b are repeated for each pair of classes, at worst this means $\frac{n_C(n_C-1)}{2}$ number of times.

Operation 2 only occurs for non-leaf nodes. A property of the binary tree is that it contains exactly $\frac{n_n-1}{2}$ non-leaf nodes (and n_n is always odd).

Hence, the total number of times an expression is evaluated at worst:

$$\begin{aligned} \mathcal{C} &= n_n\left(\frac{n_C(n_C-1)}{2}\right)(n_{\mathcal{D}} n_g n_{cs} + 2n_{\mathcal{D}}) + \frac{n_n-1}{2} n_{\mathcal{D}} \\ &= n_{\mathcal{D}} \frac{n_n n_C(n_C-1)(n_g n_{cs}+2) + (n_n-1)}{2} \end{aligned} \tag{4.4}$$

This gives a rough indication of which variables are important. In particular n_C being squared, it is an important factor. However, it does not tell the whole story. Indeed, when we are not considering the worst case, a huge factor of time complexity is how long it takes for symbolic classification to stop, which in most cases is equivalent to how long it takes to converge to the best possible solution. This highly depends on the number of input variables n_x and operators n_O that are actually the most

influential factors on time complexity (because the worst case is always far from being reached). The more there are, the longer it takes to reach the best possible solution. More specifically, since the candidate solutions are sequences of operators and variables picked randomly among the set of input variables and operators, for an expression of length l_{avg} there are $(n_x + n_O)^{l_{avg}}$ possibilities. This is not exactly true since a candidate solution cannot end with an operator, but it gives a good idea of the growth of the size of the space of possible solutions according to n_x and n_O. The paper Virgolin and Pissis, 2022 shows that symbolic regression (and by extension, symbolic classification) is an NP-hard problem.

Another important factor that is not taken into account in the computing of $\mathcal{C}$, despite being very influential on time complexity, is the average length l_{avg} of candidate solutions. The longer the expression, the longer it takes to compute it. The *parsimony coefficient* (see Section 4.2.2.3 on page 58) plays an important role in minimizing l_{avg}. Setting it to 0 causes the computational time to increase drastically.

One more factor simply is the nature of the input operators. For instance, it takes more time to compute the logarithm of a value than to add two values together. It is dependent on the inputs and cannot be estimated in a general case.

Finally, luck plays a role in how fast symbolic classification converges (in the initial candidates generation and in the mutations).

One might also wonder about space complexity. No analysis has been made because on all use cases considered as of now, no limit has ever been reached (despite sometimes using laptops with 8GB RAM). The dataset being loaded in memory prior to the algorithm starting, $n_{\mathcal{D}}$ impacts space complexity. At the beginning of a generation, during symbolic classification, n_{cs} candidate solutions of size l_{avg} are generated simultaneously. However, candidate solutions that are not used in the next generation are dynamically discarded in order to free space, leading to no accumulation.

For more information on time and space complexity of genetic algorithms, see Oliveto and Witt, 2015, Lissovoi and Oliveto, 2019 and Ankenbrandt, 1991.

4.2.3 DT4X Properties

This section aims to show what properties the resulting decision tree from DT4X possesses. These properties are derived from the way DT4X builds the decision tree. The first three properties are inherent because the algorithm was designed with these properties in mind.

First, we need to define the core concept of *data-based analytical redundancy relation* (data-based ARR). Diagnosis indicators found by DT4X have the same properties as model-based ARR but they are computed from a dataset $\mathcal{D}$. Hence, we define the specific diagnosis indicators found along the DT and name them *data-based ARR*. Consequently, data-based ARR have a validity domain limited to the dataset $\mathcal{D}$.

Definition 47 (Data-based ARR for a dataset $\mathcal{D}$)**.**
Consider a dataset of samples $\mathcal{D} = \{(x, l)\}, x \in \mathcal{R}, l \in C$ and $C_0 \in C$ the class of nominal samples. A relation of the form $\mathbf{d}(x', \dot{x}', \ddot{x}', ...) = r$, with input x', a subvector of x, and output r, a scalar named residual*, is a* data-based ARR *for the dataset $\mathcal{D}$ if, for all x such that there exists a sample $(x, C_0) \in \mathcal{D}$, it holds that $r = 0$.*

Also, some of the properties rely on being in an ideal scenario.

Definition 48 (Ideal Scenario)**.** *An ideal scenario requires the 3 following conditions:*

- *the data is perfectly clean, i.e. it contains no noise;*

- *the operators given to DT4X are sufficient to find a diagnosis indicator to separate a pair of classes when it exists.*
- *enough time is given to symbolic classification to converge to a solution if it exists.*

The main hypothesis on the ideal scenario is that, if an expression of the observable variables can split the two classes thanks to symbolic classification, it will be found by the algorithm in finite time.

Such an ideal case does not exist. However, it is possible to get close to it. Enough samples and enough generations can erase this luck factor and the perfect data can be obtained by working with non noisy systems in discrete environments (for instance with logic circuits as considered in Section 5.1.2 on page 74).

4.2.3.1 Inherent Properties

Theorem 4.2.1 (DT4X diagnosis indicators are Residual Generators)**.** *The diagnosis indicators found by DT4X are residual generators for their training data.*

Proof. As explained in Section 4.2.2.2 on page 57, symbolic classification looks for diagnosis indicators that can discriminate some faulty cases from nominal cases. In an ideal scenario, when symbolic classification stops because of reaching a perfect fitness, the found diagnosis indicator has all the properties of a data-based ARR as defined in Definition 47 on the facing page. And, by definition, data-based ARRs are residual generators for their dataset. QED

Since model-based residual generators are also valid on clean data, when data is *clean enough* and representative of the system use cases, the found data-based ARRs are the same as the model-based ARRs. In practice, this is the case for systems whose data is generated through simulation or for systems were variables can only take a finite number of values such as logic circuits. Having data that is *clean enough* is very hard to define in general and is dependent on systems and measurement equipment.

Theorem 4.2.2 (Data-based Diagnosability for a dataset $\mathcal{D}$)**.** *Diagnosability of the studied system with respect to a dataset $\mathcal{D}$ can be deduced from the leaves of the tree produced by DT4X. All faulty classes whose samples share a leaf node with nominal samples are non-detectable. All faulty classes whose samples end up in the same leaf are non-isolable from each other.*

Proof. In an ideal scenario, classes whose samples end up alone in a leaf are isolable since there exist diagnosis indicators that can separate them from the other classes (those diagnosis indicators can be found on the path from the root node towards the leaf in question). Classes whose samples end up in the same leaf as nominal samples are non-detectable because symbolic classification can not find any diagnosis indicator able to separate them and did not converge, meaning that there exists no expression of the observable variables that can split these classes samples from the nominal samples (see Definition 48 on the preceding page). This reasoning can also be applied to say that classes whose samples end up in the same leaf are non isolable from each other. These properties are assessed for the considered dataset $\mathcal{D}$ used for training DT4X. QED

Theorem 4.2.3 (Predicting the Class of an Unknown Sample)**.** *The output tree of DT4X is able to predict the class of an unlabeled sample.*

	nominal	f_1	...	f_{n_C}
$\mathbf{d}_{n_0}$	0	1	...	0
$\mathbf{d}_{n_1}$	0	?	...	1
...	...	...	...	...
$\mathbf{d}_{n_{nn}}$	?	?	...	1

TABLE 4.3: Example of Signature Matrix Inferred from the Decision Tree

Predicting the class of an unknown sample can be done by evaluating the diagnosis indicators with the sample features, i.e. observable variable values, and then following the corresponding arrows on the edges of the tree, starting from the root node. Ending up in a leaf with label l means that this sample is predicted as being of class l. A leaf containing multiple classes can be labeled as *either one of the classes*. It is not possible to say which one, since they are non-isolable according to Theorem 4.2.2 on the preceding page.

4.2.3.2 Unicity of Diagnosis Indicators on a Path

Theorem 4.2.4 (Unicity of diagnosis indicators on a path)**.** *On a given path $\mathcal{P}$ of the DT4X decision tree, each diagnosis indicator is unique. This means that along a given path, it is impossible to encounter twice the same diagnosis indicator to partition the samples of a node $n_1 \in \mathcal{P}$ and to partition the samples of a node $n_2 \in \mathcal{P}$, i.e.* $\forall n_1, n_2 \in \mathcal{P}, \mathbf{d}_1 = \mathbf{d}_2 \implies n_1 = n_2$.

Proof. Let us consider a path $\mathcal{P}$ in a DT4X decision tree. Let us consider two random diagnosis indicators along this path, the first encountered (starting from the root node) being $\mathbf{d}_1$ and the second $\mathbf{d}_2$, respectively partitioning the samples of node n_1 and n_2. Let us call $\mathcal{D}_1$ and $\mathcal{D}_2$ respectively the datasets associated with n_1 and n_2, as illustrated in Figure 4.18 on the next page. Let us call $\mathcal{D}_{1_l}$ the subset of $\mathcal{D}_1$ so that $\forall x \in \mathcal{D}_{1_l}, \mathbf{d}_1(x) = 0$. Idem, let us call $\mathcal{D}_{1_r}$ the subset of $\mathcal{D}_1$ so that $\forall x \in \mathcal{D}_{1_r}, \mathbf{d}_1(x) = 1$. By construction of the tree, either $\mathcal{D}_2 \subseteq \mathcal{D}_{1_l}$ or $\mathcal{D}_2 \subseteq \mathcal{D}_{1_r}$.

Now, we need to go back to how symbolic classification works. In DT4X, symbolic classification is used on two balanced classes with at least one element each. 90% of those samples need to be predicted correctly by the obtained solution for it to be considered a diagnosis indicator (see Section 4.2.2.3 on page 58). This means that at least one element of each class must be predicted correctly. This implies that, for a valid diagnosis indicator, such as $\mathbf{d}_2$ in our case, $\exists\ x_0 \in \mathcal{D}_2, \mathbf{d}_2(x_0) = 0$ and $\exists\ x_1 \in \mathcal{D}_2, \mathbf{d}_2(x_1) = 1$.

If $\mathcal{D}_2 \subseteq \mathcal{D}_{1_l}$, $\mathbf{d}_2(x_1) = 1$ and $\mathbf{d}_1(x_1) = 0$ because $x_1 \in \mathcal{D}_{1_l}$; thus $\mathbf{d}_1 \neq \mathbf{d}_2$. Similarly, if $\mathcal{D}_2 \subseteq \mathcal{D}_{1_r}$, $\mathbf{d}_2(x_0) = 0$ and $\mathbf{d}_1(x_0) = 1$ because $x_0 \in \mathcal{D}_{1_r}$; thus $\mathbf{d}_1 \neq \mathbf{d}_2$. QED

4.2.3.3 Necessary and Sufficient Fault Signature Matrix

From the way the training data is split by the diagnosis indicators found in nodes, it is possible to fill the fault signature matrix SM. Such a matrix is shown in Table 4.3. 0 means that $\mathbf{d}(x) = 0$ for all x in the class. 1 means that $\mathbf{d}(x) \neq 0$ for all x in the class. ? means that either we do not know what $\mathbf{d}(x)$ is equal to or it can be both 0 and not 0 depending on the data sample. See Section 5.1.1.1 on page 69 for an example of what this matrix can look like for a specific system.

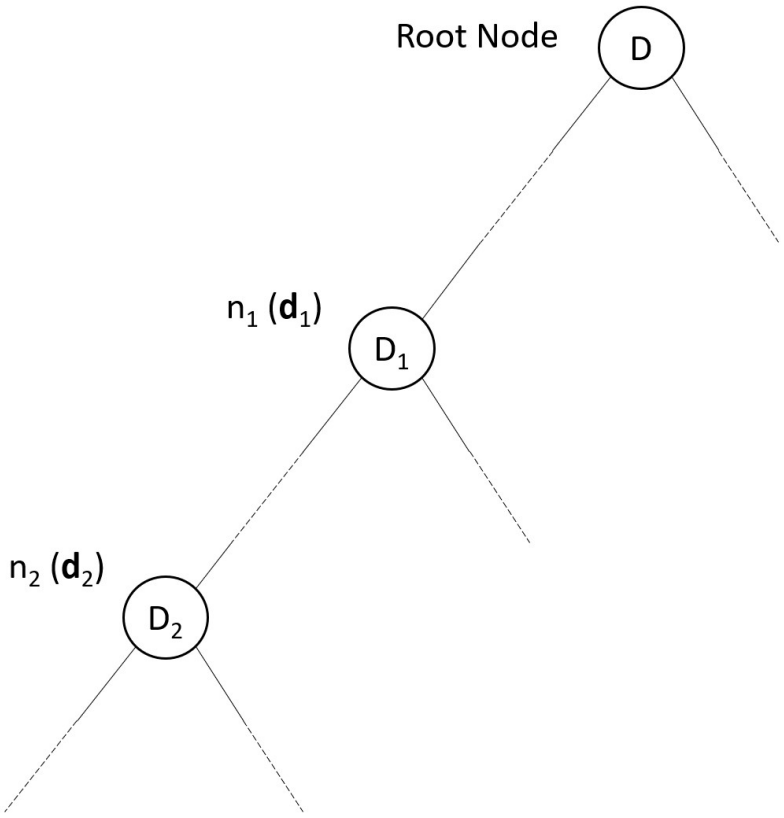


FIGURE 4.18: Simplified View of the Tree Obtained by DT4X (with $\mathcal{D}_2 \subseteq \mathcal{D}_{1_l}$)

Theorem 4.2.5 (Sufficient Fault Signature Matrix)**.** *The signature matrix that can be established from the output tree of DT4X is sufficient to obtain the maximal (data-based) diagnosability of the system.*

Proof. The signature matrix is equivalent to the tree in terms of diagnosability (it is just a way to present it in a more readable way than the tree itself). According to Theorem 4.2.2 on page 63, we know that the tree contains sufficient information to obtain maximal (data-based) diagnosability of the system. QED

However, there is more than only the necessary information to obtain maximal diagnosability. Indeed, Figure 4.19 on the following page shows a counter-example.

This is an example of a phenomenon that may happen in a tree from DT4X. Each node contains the class numbers. The two orange nodes contain the same classes but two different diagnosis indicators are found. This may happen, for instance, if the pairs for symbolic classification are $(1, 2)$ and $(2, 1)$. The corresponding two diagnosis indicators would be different (because one would be zero for class 1 and the other for class 2) despite being redundant in terms of diagnosability information. This proves that the signature matrix obtained from DT4X can have more information than necessary for maximal (data-based) diagnosability.

4.2.3.4 Bounded Amount of Data Required to Train

It is conjectured that the amount of data required to train DT4X has a ceiling. This means that beyond a certain amount, DT4X performances do not improve. The reasoning behind this conjecture is that once the model-based diagnosis indicators of the

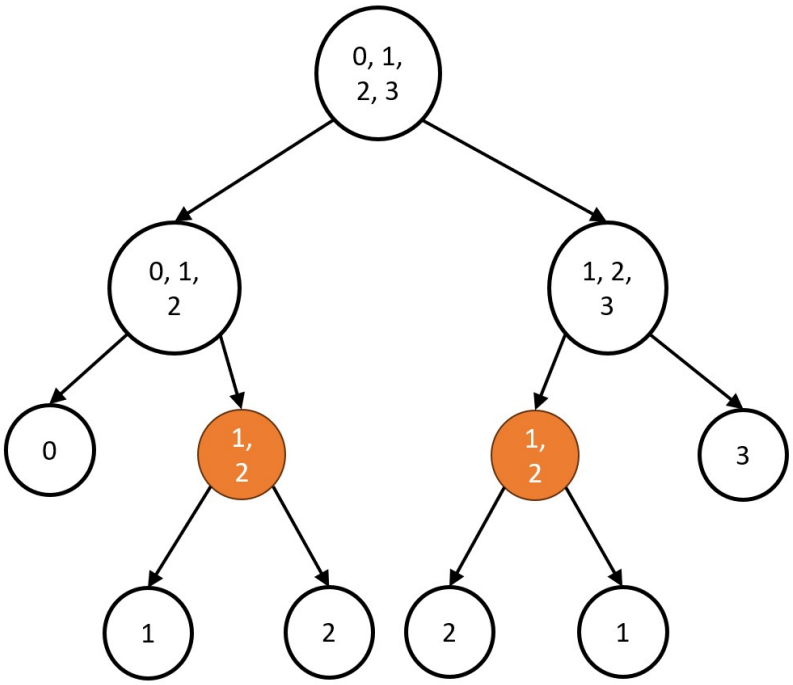


FIGURE 4.19: Tree from DT4X that Contains more Information than Necessary

system are found by DT4X, it reaches maximum performances. After that, it cannot find better expressions and thus, cannot improve. However, it seems very complex to prove that DT4X always finds model-based ARRs, even in an ideal scenario. Note also that model-based ARRs can be taken as reference only in the case of a "perfect" model, which hardly happens for dynamic systems. However, it can be easily true for static systems like the polybox (cf. Section 5.1.1 on page 69) and logic circuits (cf. Section 5.1.2 on page 74).

4.2.3.5 Kernel Intersection of Data-Based ARRs from DT4X is Included in the Kernel Intersection of Model-Based ARRs

Definition 49 (Kernel of a diagnosis indicator)**.** *The kernel of a function f defined on a set I is defined as $Ker(f) = \{x \in I, f(x) = 0\}$. Thus, the kernel of a diagnosis indicator $\mathbf{d}_{n_i}$ on $\mathcal{D}$ is defined as $Ker(\mathbf{d}_{n_i}) = \{x \in \mathcal{D}, \mathbf{d}_{n_i}(x) = 0\}$.*

Theorem 4.2.6 (Kernel Inclusion)**.** *In an ideal scenario with the hypothesis that all fault classes are detectable and that model-based ARRs are built from an accurate model of the system:*

$$\bigcap_{\mathbf{d} \in \mathbf{D}} Ker(\mathbf{d}) \subseteq \bigcap_{arr \in \mathcal{R}} Ker(\texttt{arr}) \tag{4.5}$$

with $\mathbf{D}$ the set of all diagnosis indicators found when training DT4X and $\mathcal{R}$ the set of model-based ARRs for the studied system.

Proof. Let us call C_{nom} the class of C that contains the nominal samples. First, we show that $C_{nom} = \bigcap_{\mathbf{d} \in \mathbf{D}} Ker(\mathbf{d})$ (i). Then, we show that $C_{nom} \subseteq \bigcap_{arr \in \mathcal{R}} Ker(\texttt{arr})$ (ii).

(i) In an ideal scenario, the diagnosis indicators found by DT4X are data-based ARRs (Theorem 4.2.1 on page 63). By definition of a data-based ARR, $\forall\, x \in C_{nom}$, $\forall\, \mathbf{d} \in \mathbf{D}, \mathbf{d}(x) = 0$. Thus, $C_{nom} \subseteq \bigcap_{\mathbf{d} \in \mathbf{D}} Ker(\mathbf{d})$. Let us consider $x \in \bigcap_{\mathbf{d} \in \mathbf{D}} Ker(\mathbf{d})$. $\forall$

$\mathbf{d} \in \mathbf{D}, \mathbf{d}(x) = 0$. This means that there is no diagnosis indicator that can detect x. Either x is nominal or x belongs to an undetectable class. With the hypothesis that all classes detectable, we can deduce that $x \in C_{nom}$. Consequentially, $\bigcap_{\mathbf{d}\in\mathbf{D}} Ker(\mathbf{d}) \subseteq C_{nom}$ and so $C_{nom} = \bigcap_{\mathbf{d}\in\mathbf{D}} Ker(\mathbf{d})$.

(ii) Let us call X_{nom} the set of samples coherent with the normal behavior represented by the system model. By definition of a model-based ARR and using the hypothesis that all faults are detectable, $X_{nom} = \bigcap_{arr\in\mathcal{R}} Ker(\texttt{arr})$. All samples from C_{nom} are measured on the system in nominal conditions, meaning that $C_{nom} \subseteq X_{nom}$, because ARRs have been computed from an accurate model of the system. Therefore, $\bigcap_{\mathbf{d}\in\mathbf{D}} Ker(\mathbf{d}) \subseteq \bigcap_{arr\in\mathcal{R}} Ker(\texttt{arr})$. QED

4.3 Conclusions

4.3.1 Summary

DT4X is a data-driven algorithm that uses meta-knowledge from model-based diagnosis in order to perform diagnosis of a system whose model is unknown or only partially known. DT4X trains a decision tree to diagnose a system. It:

- trains a multivariate decision tree, the expressions in each node are diagnosis indicators that express relations between observable variables in a nominal case.
- is explicable in the sense that the obtained relations are data-based ARRs that have physical meaning in the system, but it also gives an explanation of what the diagnosability of the system is by looking at class distribution in the leaves of the tree.
- uses symbolic classification to find diagnosis indicators without inputting an expected analytical form.
- incorporates meta-knowledge from model-based diagnosis by tuning the classification function of symbolic classification to constrain it to look for diagnosis indicators that are null for nominal cases and not null for at least a fault class.

DT4X is designed in a way that allows expert knowledge to be taken advantage of. Indeed, the choice of relevant operators and system constants can help DT4X converge faster.

However, there are some limitations to using DT4X:

- A labeled dataset has to be available. It has to contain classes from all faulty faults (not necessarily in a big amount though, and not as much as nominal cases).
- It is based on a genetic algorithm that is inherently random and can take a variable, and hard to estimate, time to converge.
- Hyper-parameters can be hard to fine-tune. The value of ϵ and the parsimony coefficient have a huge impact on the outcome of the training and there is no rule on how to identify a good value for them.

4.3.2 Perspectives

4.3.2.1 More Expert Knowledge

Future works could focus on inputting pre-made expressions in symbolic classification. Indeed, some model knowledge of the system might be easy to obtain and it would be interesting to be able to take advantage of it. Let us say that we know the equation of a simple component of the system. In the current state of DT4X, it is not possible to use this knowledge. One way to use it would be to initialize some expression trees with this component equation during symbolic classification. It would save time for symbolic classification.

4.3.2.2 Automatic Fitting of Hyper-Parameters

As mentioned previously, finding good hyper-parameters can be a very complex task. Here are two ideas to solve this issue.

First, the parsimony coefficient $\mathbf{p}$ is used in this way:

$$fit_{final} = \mathbf{p} \times len + fit_{initial} \tag{4.6}$$

with len the len of the expression, $fit_{initial}$ the fitness before applying the parsimony coefficient, computed as described in Section 4.1.2 on page 46, and fit_{final} the final fitness, post application of the parsimony coefficient. Hence, the effect of $\mathbf{p}$ is very dependent on the order of magnitude of len and $fit_{initial}$. With expressions of average length 100 and $fit_{initial}$ around $10e^{-6}$, if $\mathbf{p} \geq 10e^{-7}$ its impact overshadows the impact of $fit_{initial}$, which is not wished. Hence, the idea is to have an adaptive parsimony coefficient that scales with the order of magnitude of $fit_{initial}$. It could even vary online according to the $fit_{initial}$ of the best candidate solution c_{best}.

Second, a similar reasoning could perhaps be applied to ϵ. Indeed, there are algorithms able to automatically find the right threshold for a residual generator with some labeled data. For instance, CUSUM tests (Riaz, Abbas, and Does, 2011) are able to detect small variations in signals and could be used to find the right threshold to separate two different sections of a residual generator signal.

4.3.2.3 Refitting Following Concept Drifts

The `refit` function allows to retrain part of the tree. This could potentially be used to retrain a section of the tree that would not be relevant anymore because of a concept drift in the dataset (for instance the studied system deteriorating and not behaving exactly the same, or a component being replaced).

Chapter 5

DT4X Applications

This chapter goal is to showcase the performance of DT4X on four systems. First, a very simple static system, the polybox, used to illustrate how DT4X is applied and what the results are in an ideal case. Next, the full subtractor, a logic circuit, is presented. Then, two dynamic systems, the water tanks and the 3D printer are tested. Finally, a look back at what worked and what did not is taken and a few perspectives on how to improve DT4X are discussed.

5.1 Application to Static Systems

5.1.1 Polybox

5.1.1.1 System Description

At first, let us begin with a fully solved and very simple static system: the polybox (De Kleer and Williams, 1987). The point is to test DT4X on a simple case. If it does not work on this case, no need to test it on a dynamical, noisy, high order, real system.

The polybox is a fictional static system that contains five components: M1, M2, M3, A1 and A2. These components are connected as shown on Figure 5.1. The M1, M2 and M3 components are multipliers (the output equals the product of the inputs) and the A1 and A2 components are adders (the output equals the addition of the inputs).

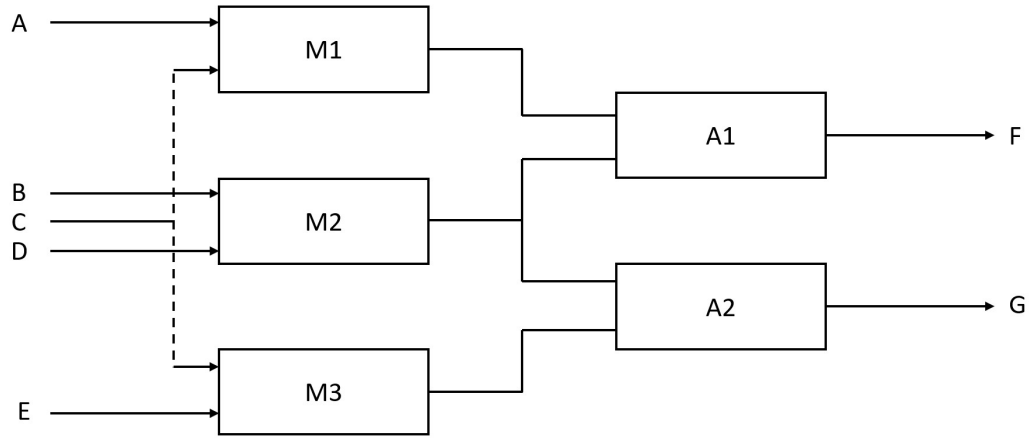


FIGURE 5.1: The Polybox

Each component can malfunction, meaning the component does not produce the correct output according to the inputs. Thus, there are five possible faults in this

system. For the sake of simplicity, the fault associated with component M1 is called f_{M1} and *idem* for the other components.

There are seven observable variables in this system: A, B, C, D, E, F and G. The pairs in the dataset are of the shape (x, l) with $x = (A, B, C, D, E, F, G) \in \mathbb{N}^7$ and $l \in C$ with C the set of possible diagnoses (C is detailed below). For now, only non ambiguous states (see Definition 6 on page 4) are considered as classes.

In the context of this study, two experiments have been made. One with single faults only, meaning that only one component can be faulty. The other experiment is with double faults (two components faulty at the same time) as well as single faults. It can be noted that cases with triple, quadruple and all faults are very similar to the double faults case, and work the same and thus are not tackled.

If only single faults are considered, $C = \{nominal, f_{M1}, f_{M2}, f_{M3}, f_{A1}, f_{A2}\}$.

5.1.1.2 Results

A dataset of 1024 nominal samples and 1024 faulty samples, each being of one fault type, has been randomly generated. A fault or a component malfunction is defined as an output from this component that is different from the expected value. Thus, a faulty component outputs the expected value plus a random modifier in $\{-3, -2, -1, 1, 2, 3\}$. Then, this dataset is randomly split between a training set with 1434 samples and a testing set with 614 samples. The training set is fed to DT4X with default hyper-parameters and the output decision tree is shown in Figure 5.2. The accuracy for this decision tree on the test set is 80.78%.

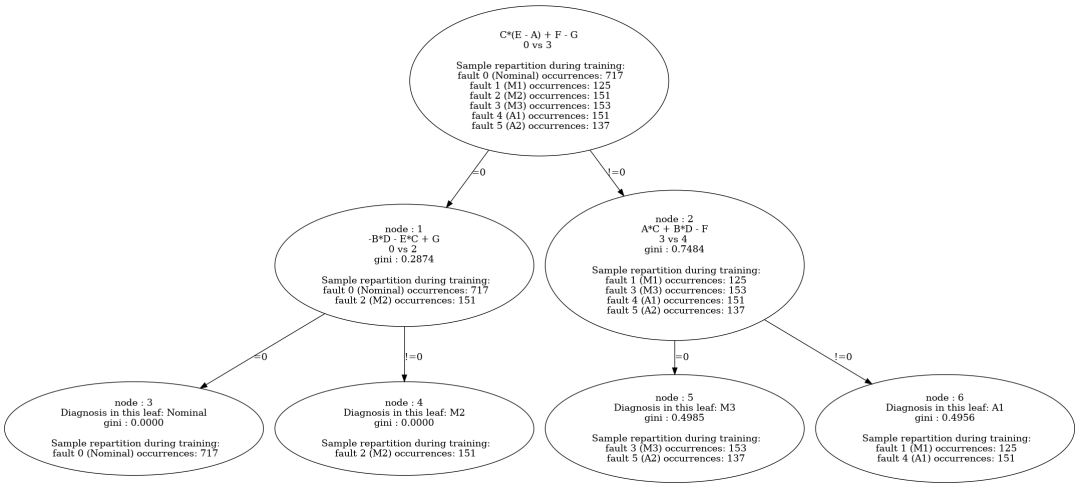


FIGURE 5.2: Single Fault Polybox DT4X Decision Tree

The confusion matrix of this decision tree (computed on the test set) is shown in Figure 5.3 on the facing page.

5.1.1.3 Comparison with Model-Based Results

In order to evaluate the quality of the found expressions, let's compute them using structural analysis (and the model of the system). The system equations are the following:

$$e_1 : AC + f_{M1} = x \tag{5.1}$$

$$e_2 : BD + f_{M2} = y \tag{5.2}$$

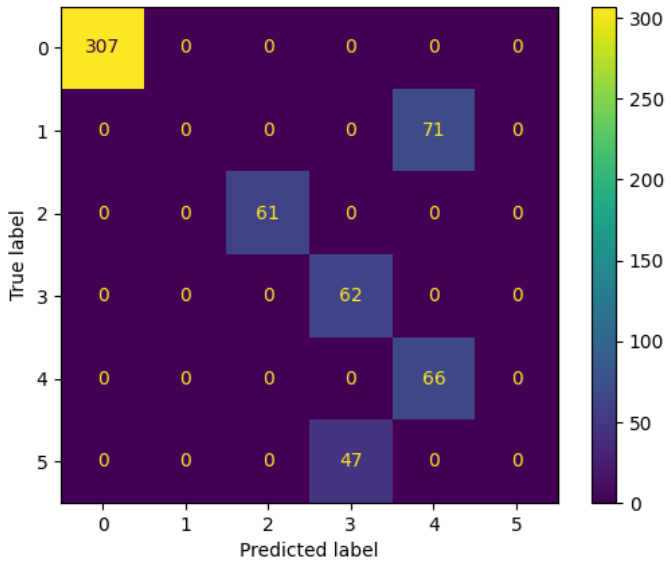


FIGURE 5.3: Confusion Matrix of the Single Fault Polybox Diagnosis Tree

$$e_3 : CE + f_{M3} = z \tag{5.3}$$

$$e_4 : x + y + f_{A1} = F \tag{5.4}$$

$$e_5 : y + z + f_{A2} = G \tag{5.5}$$

With x, y and z being the internal states of the system (non-observable inner variables).

Using the fault diagnosis toolbox we can display the structural model of the polybox (Figure 5.4). Either using this toolbox also, or using the Dulmage-Mendelsohn

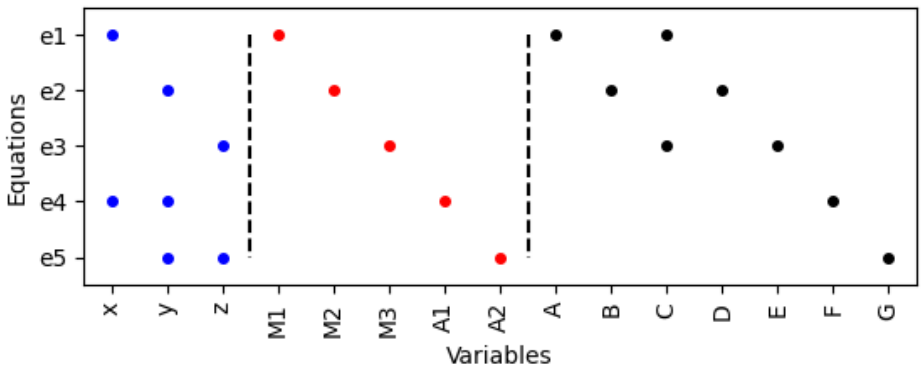


FIGURE 5.4: Structural Model of the Polybox

decomposition (Dulmage and Mendelsohn, 1958), we compute MSO sets of equations. The MSO sets for this system are:

$$MSO_1 = \{e_1, e_2, e_4\} \tag{5.6}$$

$$MSO_2 = \{e_2, e_3, e_5\} \tag{5.7}$$

$$MSO_3 = \{e_1, e_3, e_4, e_5\} \tag{5.8}$$

Finally, for each MSO set, we can compute the associated residual generator by eliminating non-observable variables in the equation set (and considering only the nominal case).

$$r_{MSO_1} = AC + BD - F \tag{5.9}$$

$$r_{MSO_2} = BD + EC - G \tag{5.10}$$

$$r_{MSO_3} = AC - EC + G - F \tag{5.11}$$

The fault support of MSO_1 is $\{f_{M1}, f_{M2}, f_{A1}\}$ and the fault support of MSO_2 is $\{f_{M2}, f_{M3}, f_{A2}\}$. Hence, these two MSO sets are enough to obtain full detectability of the faults. Since the fault support of MSO_3 is $\{f_{M1}, f_{M3}, f_{A1}, f_{A2}\}$, we can deduce that f_{M1} and f_{A1} are non-isolable, and idem for f_{M3} and f_{A2}. This also means that maximal isolability for this system can be obtained from two different MSO sets only (no matter which ones).

The DT4X decision tree in Figure 5.2 on page 70 shows that the expressions found by DT4X are identical to the three residual expressions from model-based diagnosis. The classes present in the leaf nodes during training also reflect the diagnosability of the system. Multiple runs of DT4X with different random seeds for symbolic classification generate different trees. For this same data, sometimes the generated tree ends up having the same expressions in node 1 and node 2. This reflects the fact that only two residuals are necessary and sufficient to obtain maximal diagnosability for this system.

We can represent the fault supports (computed from the model) using the fault signature matrix (Table 5.1). We can also do the same for the expressions found with

	nominal	f_{M1}	f_{M2}	f_{M3}	f_{A1}	f_{A2}
MSO_1	0	1	1	0	1	0
MSO_2	0	0	1	1	0	1
MSO_3	0	1	0	1	1	1

TABLE 5.1: Fault Signature Matrix for the Single Fault Polybox (computed from the model)

DT4X, by filling the signature matrix as described in Section 4.2.3.3 on page 64. The obtained matrix is presented in Table 5.2 on the next page. Since the expression of node 0 corresponds to MSO_3, it makes sense to find the same fault signature. Node 1 corresponds to MSO_2 and node 2 to MSO_3 and their respective signatures could match the ones of the MSO sets. However, we do not have enough information to tell. The interesting part is that this information we are lacking is not mandatory to obtain maximal diagnosability for this system. Indeed, by looking at Table 5.1 that f_{M1} and f_{A1} have the same signature, meaning that they are not isolable from each other. The same can be said of f_{M3} and f_{A2}. Other than those two pairs, the decision tree managed to isolate all the classes. Thus, the obtained decision tree has achieved maximal possible diagnosability of the polybox, despite having less overall information (the **?** cells of Table 5.2 on the next page).

5.1.1.4 Comparison with Other Machine Learning Algorithms

In order to evaluate the performance of DT4X and compare it with other algorithms we use the f1-score. This metric expresses both precision and recall in a single metric and can also take into account the imbalance of classes. In order to compute it, we determine per-class f1-score and then take the average of those scores, weighted by

	nominal	f_{M1}	f_{M2}	f_{M3}	f_{A1}	f_{A2}
node 0	0	1	0	1	1	1
node 1	0	?	1	?	?	?
node 2	?	1	?	0	1	0

TABLE 5.2: Fault Signature Matrix for the Single Fault Polybox (computed from DT4X tree)

true class size (class size in the dataset, not in the predictions). The expression of the f1-score of class C_i is:

$$f1score(C_i) = \frac{2TP}{2TP + FP + FN} \tag{5.12}$$

with TP the true positives (correctly predicted samples of C_i), FP the false positives (sample from an other class predicted as belonging to C_i), FN the false negatives (samples from C_i predicted as from an other class). The expression of the f1-score for an algorithm $\mathbf{A}$ is given by:

$$f1score(\mathbf{A}) = \frac{1}{|\mathcal{D}|} \sum_{C_i \in \mathcal{C}} |C_i| f1score(C_i) \tag{5.13}$$

A high f1-score means that recall and precision are high, but also that there is no imbalance between both.

The accuracy of DT4X decision tree compared to other default scikit-learn implementations of common machine learning algorithms are shown in Table 5.3. The

Algorithm	Scoring Time (s) 614 samples	Accuracy (%)	f1-score (%)
DT4X	0.04	80.78	74.25
sklDT	0.00	46.91	46.02
sklRF	0.01	46.09	38.41
sklLR	0.00	50.00	33.33
sklNB	0.00	49.67	34.17
sklSVM	0.02	50.16	33.69
sklKNN	0.01	48.05	37.43

TABLE 5.3: Single Fault Polybox Results

double fault polybox results are presented in Section B.1.1 on page 117 of Appendix B on page 117.

DT4X outperforms other machine learning algorithms on this polybox case. However, its accuracy is far from 100%. This is because, as mentioned before, some faults are not isolable from one another (this is known from model-based diagnosis). For instance, f_{M1} and f_{A1} have the very same signature. Perhaps, this is also the reason for which the standard machine learning algorithms are struggling. Thus, we ran the same tests while considering the non-isolable classes as the same, merging them together in one class that represents an ambiguous state. The results are presented in Sections B.1.2 on page 117 and B.1.3 on page 118 of Appendix B on page 117. In that context, the main thing to notice is that DT4X has perfect or near-perfect accuracy. The only few samples that are misclassified in the double fault case are ones that behave like data from other classes. This perfect accuracy is due to finding the exact

expressions that the physical model would have allowed to compute. Obviously, with noise and imperfect and hidden data, an accuracy this high can not be reached (but this is also true with access to the full physical model of the system).

Overall, DT4X takes longer to train (around a thousand times longer than the average for the machine learning algorithms), longer to test on data (which is way more important than training that only has to be done once, hence why it is shown in the tables), and has way higher accuracy.

In the node 2 of Figure B.3 on page 119 there is an interesting phenomenon. The expression is made of two factors, with the right one being a diagnosis indicator for the data, while the left one is simply never null (because it is the diagnosis indicator from the first node and was evaluated as not null for the data in node 2). This left factor does not affect whether the expression evaluated on the data is null or not, because it is never null. This case is rather unlikely because it means that this expression has to be randomly found before only the right part. Because symbolic classification is based on a genetic algorithm, itself partly reliant on randomness, this scenario can occur. It does not affect the performances of the tree. Also, this is more likely to occur when working with natural numbers (otherwise, the left factor could make the expression reach below the threshold ϵ). Finally, the higher the *parsimony coefficient* is, the less likely it is to happen.

5.1.1.5 Other Variants

There is an infinite number of variants of the standard polybox shown in Figure 5.1 on page 69. We tried to run DT4X on two other kinds of polybox, in order to show that it can perform consistently on simple static systems without noise. The results are presented in Appendix B on page 117.

With the success that DT4X has on the polybox cases, it begs the question: how would it behave on real world systems ? At first, we take a look at how it performs for a very similar, yet concrete, category of systems, logic circuits.

5.1.2 Logic Circuits

5.1.2.1 Introduction

Logic circuits are physical or abstract electronic circuits designed to perform logical operations on one or more binary inputs to produce one or more binary outputs. These circuits are the building blocks of digital systems and are used extensively in computers, calculators, communication devices, and various other electronic devices.

In a logic circuit, binary values (0 and 1) represent logical states (false and true, respectively). The fundamental components of logic circuits are logic gates, which are electronic devices that perform basic logic operations such as AND, OR, NOT and XOR.

In this manuscript, we only deal with combinational logic circuits (Jain et al., 1997), meaning circuits that produce outputs solely based on the current input values, without considering past inputs or outputs (as opposed to sequential logic circuits, Jahanirad, 2019). Combinational logic circuits are used to perform specific functions like addition, subtraction, multiplication, and comparison.

Many works tackle fault diagnosis of combinational logic circuits (Lu et al., 2003; Fujiwara and Toida, 1982; Smith et al., 2005). However, to the best of our knowledge, no work looks for diagnosis indicators built with logic operators. Applying DT4X to a logic circuit example shows that it is able to find diagnosis indicators in which the

operators are logic gates, leading to a new kind of ARRs named data-based logic ARRs.

Definition 50 (Logic ARR for a dataset $\mathcal{D}_{01}$)**.** *Consider a dataset of samples* $\mathcal{D}_{01} = \{(x, l)\}, x \in \{0,1\}^n, l \in C$ *and* $C_0 \in C$ *the class of nominal samples. A relation of the form* $\mathbf{d}_{01}(x', \dot{x}', \ddot{x}', ...) = r$*, with input* x'*, a subvector of* x*, and output* r*, a scalar named* residual*, is a* data-based logic ARR *for the dataset* $\mathcal{D}_{01}$ *if, for all* x *such that there exists a sample* $(x, C_0) \in \mathcal{D}_{01}$*, it holds that* $r = 0$*, and all operators of* $\mathbf{d}_{01}$ *are logic gates.*

As a consequence, a data-based logic ARR is a particular case of data-based ARR where all operators are logic gates.

Let us take a look at the full subtractor example.

5.1.2.2 System Description

The logic circuit example we consider is the full subtractor, shown on Figure 5.5. This system computes $D = A - B$ with A and B in $\{0,1\}$ and D the output difference. If there is a carry over, it is outputted by C_{out}. It also accepts a potential previous carry over that is inputted as C_{in}. Thus, the five observable variables in this system are A, B, C_{in}, C_{out} and D. Each component (= logic gate) of the system can be faulty. A fault is modeled as the output being fixed to a value (0 or 1), meaning it always outputs the same value independently of the inputs. This is called a stuck-at fault and is the most common way to represent a fault in combinational logic circuits (Lu et al., 2003). Thus, the set of possible diagnosis classes $C = \{nominal, f_{XOR1}, f_{XOR2}, f_{NOT1}, f_{AND1}, f_{NOT2}, f_{AND2}, f_{OR}\}$.

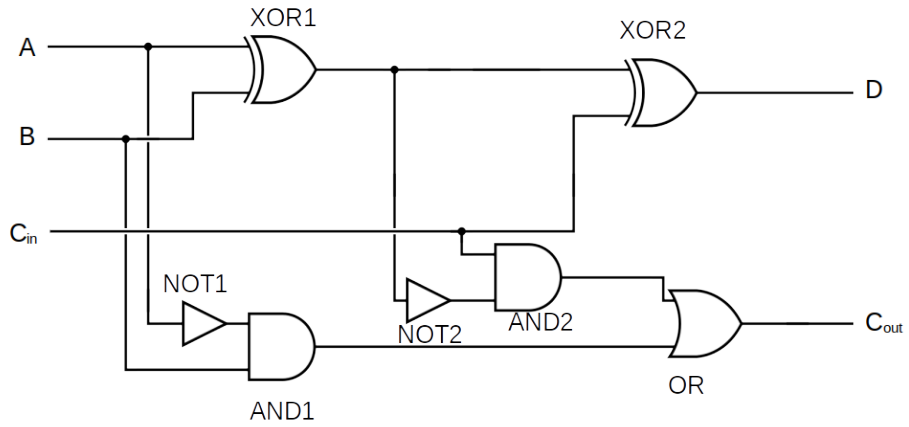


FIGURE 5.5: The Full Subtractor

A dataset with 125000 nominal samples and 125000 faulty samples is generated by simulating the subtractor in Python. For each faulty sample, exactly one random component malfunctions.

5.1.2.3 Masked Faults and Preprocessing

In a logical framework, among the generated faulty samples, a lot of them are identical to nominal samples. This is called a masked fault.

Definition 51 (Masked Fault)**.** *A masked fault is a sample labeled as faulty that is identical to a sample that is not faulty.*

For instance, an AND gate is stuck at outputting 0 and receives $(0, 0)$ as inputs. It would have output 0, even if it was not faulty.

The main difference in the dataset between this use case and the polybox example is that the subtractor dataset contains a lot of masked faults. More than 50% of the faulty samples are masked faults. If DT4X were to be trained with this data, symbolic classification would never find diagnosis indicators able to isolate faulty classes from the nominal class because more than half the faulty samples are identical to nominal samples. In other words, for any faulty class f_i, there exists multiple samples x such that both (x, f_i) and $(x, nominal)$ are in the dataset. Hence, the need for preprocessing the data before running DT4X on it.

It is not possible to predict faults for samples that are identical to nominal samples. But it is not necessary either. Hence, we preselect a new subset of $\mathcal{D}$, named $\mathcal{D}_{faulty}$, that only contains faulty samples that do not appear in the nominal sample set: $\mathcal{D}_{faulty} = \mathcal{D} \setminus \{x, \exists(x, nominal)\}$. It is important to notice that this process is not specific to our use case. It can be applied to any combinatorial logic circuit. This filtering does not presuppose any knowledge about the system. However, it can only work on systems where variables can take a finite set of values[1] such as logic circuits.

On our full subtractor use case, after that filtering, only 46384 faulty samples are left.

5.1.2.4 Training and Results

Once the preprocessing is done, the data is reshuffled and split between two sets: the training set with 137108 samples and the testing set with 34276 samples. DT4X is trained with default parameters, a parsimony coefficient of 0.05 (in order to increase the likelihood of short expressions) and an operator set of basic binary operators: $[OR, XOR, AND, NOT]$. The obtained decision tree and its comparison with other machine learning algorithms can be found in Figure 5.6 on the facing page and Table 5.4 on the next page respectively. The confusion matrix is given in Figure 5.7 on page 78.

DT4X has an accuracy and a f1-score that can compete with the best machine learning algorithms. Also, it has something more. It finds diagnosis indicators, or as defined in Definition 50 on the previous page, data-based logic ARRs. For instance, let us look at the truth table of the expressions used in nodes. Table 5.5 on page 79 shows this truth table in the nominal case. The found expressions have all the properties required to be data-based logic ARRs. Indeed, they are expressions made of observable variables and logic gates that are worth 0 for nominal data. In this binary, non-noisy, framework, the data-based logic ARRs found by DT4X should meet model-based ARRs if the training dataset is representative enough.

[1] for continuous systems, an interval could be considered. But then, how to discriminate samples from non-isolable faults from samples that are "identical" to nominal ?

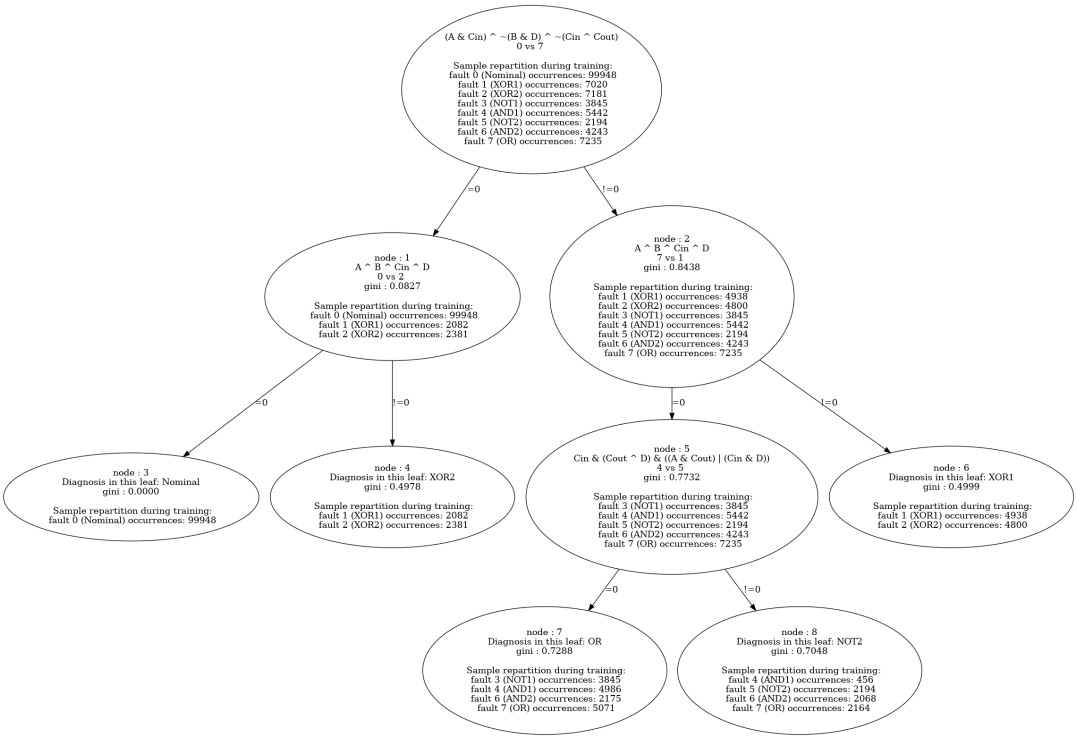


FIGURE 5.6: Single Fault Full Subtractor Decision Tree

Algorithm	Scoring Time (s) 34276 samples	Accuracy (%)	f1-score (%)
DT4X	2.48	84.37	82.09
sklDT	0.00	85.07	84.48
sklRF	0.17	84.95	83.72
sklLR	0.00	73.09	61.73
sklNB	0.02	16.80	20.56
sklSVM	28.57	84.95	83.72
sklKNN	4.34	85.18	84.30

TABLE 5.4: Single Fault Full Subtractor Results

Another interesting aspect to look at is how these logic ARRs perform (analytically) on faulty cases. Let us consider the performance of the logic ARR found in node 1 on faults f_{XOR1} and f_{XOR2} (Table 5.6 on page 79). This shows that indeed, the expression has the properties of a residual whose fault support would be f_{XOR1} and f_{XOR2}. It is sensitive to data from these faulty classes.

Now, let us study how it fares against dynamic systems, and also systems with unclean (e.g. noisy) data.

5.2 Application to Dynamic Systems

This section showcases the performances of DT4X on the water tanks and the 3D printer use cases.

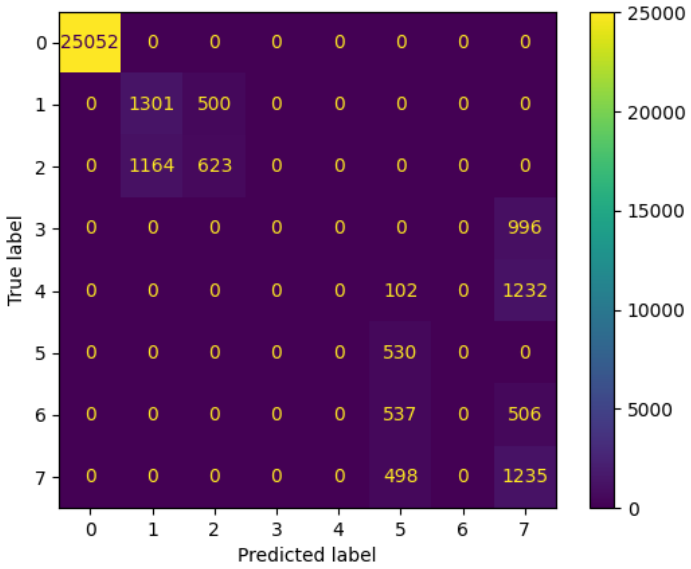


FIGURE 5.7: Confusion Matrix of the Full Subtractor Diagnosis Tree

5.2.1 Specifics about Dynamic Systems

DT4X trains from samples corresponding to specific time steps, which means it does not take into account any temporal information such as the next or previous time steps. Indeed, ideally DT4X would be used in a real-time context where the next time steps are unknown, and only a limited amount of previous time steps are known. Fortunately, there exist other ways to give information about dynamics of the system, for instance giving time derivatives or integrals of the variables.

Thus, when working with dynamic systems, derivatives (and/or integrals) of the variables are provided as input to DT4X alongside the variables themselves (Mohammadi et al., 2023). In practice, in this manuscript, we only work with derivatives and not integrals. Derivatives contain information about the system dynamics and are needed to build relevant diagnosis indicators. This implies prior knowledge of the highest derivative order required to obtain diagnosability. In the current implementation, numerical computation of the derivatives is performed for all the continuous domain observable signals of x during the data preprocessing stage and added to the feature vector given as input to DT4X.

One might consider that a more effective solution is to incorporate a derivative operator to be used in the symbolic classification phases of DT4X. It would allow candidate solutions to contain any order derivatives and to compose the derivative operator and other operators. However, when processing a sample x, this would require accessing the neighboring samples, making this solution more resource-intensive in terms of both memory and computational requirements. Also, deciding the size of the window of neighboring samples is a problem as complex as deciding the maximum derivative order (exactly equivalent, in fact).

The main issue encountered when working with derivatives is how to differentiate noisy signals. Indeed, differentiating a noisy signal outputs unusable derivatives because the noise amplitudes become much higher than the actual variations of the derivative signals themselves. Many works mention this problem and focus on solving it, either by denoising the original signal (Y.-M. Chen et al., 2016) with, for instance,

Inputs			Outputs		Diagnosis Indicators		
A	B	C_{in}	D	C_{out}	$(A\&C_{in})$ $\hat{}\neg(B\&D)$ $\hat{}\neg(C_{in}\&C_{out})$	$A\hat{}B$ $\hat{}C_{in}\hat{}D$	C_{in} & $(C_{out}\hat{}D)$ & $((A\&C_{out})\mid(C_{in}\&D))$
0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	0
1	0	0	1	0	0	0	0
1	0	1	0	0	0	0	0
1	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0

TABLE 5.5: Truth Table of Diagnosis Indicators Found by DT4X for the Full Subtractor System (in the nominal case)

Inputs			Outputs				Expression	
			f_{XOR1}		f_{XOR2}		f_{XOR1}	f_{XOR2}
A	B	C_{in}	D	C_{out}	D	C_{out}	$A\hat{}B\hat{}C_{in}\hat{}D$	
0	0	0	1	0	1	0	1	1
0	0	1	0	0	0	1	1	1
0	1	0	0	1	0	1	1	1
0	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	1	1
1	0	1	1	1	1	0	1	1
1	1	0	1	0	1	0	1	1
1	1	1	0	0	0	1	1	1

TABLE 5.6: Truth Table of Second Node Diagnosis Indicator for faults f_{XOR1} and f_{XOR2}. Once the data is filtered, the only data remaining are visible faults, meaning a faulty component outputs the incorrect value. This table is built in this specific context.

low pass filtering (Dolabdjian, Fadili, and Leyva, 2002), or by finding a way to differentiate that is robust to noise (Sterten and Furtat, 2017; Fioretti and Jetto, 1994). There is no method that works for all types of problems, and many such problems still have no solution. After trying all the noisy differentiation methods present in Van Breugel, Kutz, and Brunton, 2020, we can safely assume that no method is able to compute relevant noisy derivatives for the dynamic system studied here, the water tanks, submitted to sensor noise variance of between $2.5e^{-5}$ and $5e^{-4}$ amplitude.

5.2.2 Water Tanks

The system has been described in Section 3.4.1 on page 36. The following sections describe how a dataset is built, DT4X is used and what the results are. Then, the results are analyzed.

5.2.2.1 Dataset

A dataset of 110 simulations is generated, 10 simulations per fault type (and 10 faultless simulations). The sampling frequency is 50Hz. For the aforementioned reasons, in this dataset, no noise is injected. The dataset contains measurements of four

d_1	0.0452
d_2	0.0638
d_3	0.0591
d_4	0.0878
d_5	1.4107
d_6	2.0964

TABLE 5.7: System Constants

observable variables: y_1, y_2, y_3, y_4 and a label l corresponding to the fault type (or nominal). As we know from expert knowledge that most interactions in the system can be described with only first order derivatives, first order derivatives are computed from the observable signals, using the `gradient` method of the NumPy package, and added to the dataset.Since u_{ref} is the input of the system, it is known and is also included in the dataset. Finally, known constants of the system are given to DT4X as some form of expert knowledge injected into the algorithm. These are mostly component constants. They are recapped in Table 5.7. They are given in the form of features with a constant value. In the end, the dataset is split between a training and a testing dataset. The training dataset contains 3080088 samples of 15 variables ($u_{ref}, y_1, y_2, y_3, y_4, \dot{y}_1, \dot{y}_2, \dot{y}_3, \dot{y}_4, d_1, d_2, d_3, d_4, d_5, d_6$) and the corresponding label l. The testing dataset contains 770022 samples.

5.2.2.2 DT4X Results

DT4X is run on this dataset with $O = \{+, -, \times, /, \sqrt{}, ^2\}$. It uses default hyperparameters except a *maximum number of generations* of 35, a *parsimony coefficient* of 0.002, $X_r = 0.0001$ and $\epsilon = 0.03$. Those have been chosen according to expert knowledge about the system and fine-tuned after a few trials and errors. The output tree of DT4X is too large to be displayed here. It goes to depth 9 and consists of 58 nodes. Results in terms of accuracy and f1-score are presented in Table 5.8 and compared with other traditional machine learning algorithms. The confusion matrix of the results is given in Figure 5.8 on the next page.

Algorithm	Scoring Time (s) 770022 samples	Accuracy (%)	f1-score (%)
DT4X	373.74	99.78	99.78
sklDT	0.10	99.996	99.996
sklRF	7.30	99.997	99.997
sklLR	0.14	77.52	75.75
sklNB	0.49	77.51	78.37
sklSVM	-	-	-
sklKNN	15.10	99.998	99.998

TABLE 5.8: Water Tanks Results. The SVM did not finish training after more than 30 hours, hence why it has no value.

DT4X has a very high accuracy on the water tanks. It is close to the best algorithms but not better. In general, from all results we have seen so far, the best machine learning algorithm tends to be the random forest classifier, or at least it is always very close to the best one. This is very interesting because it allows some comparisons with DT4X that also relies on decision trees. DT4X is much slower to

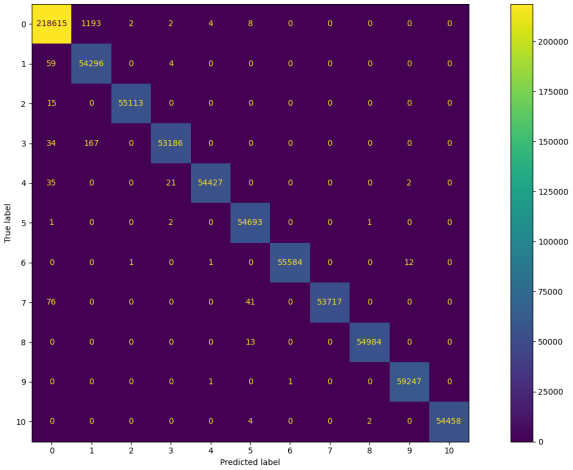


FIGURE 5.8: Confusion Matrix of DT4X for the Water Tanks

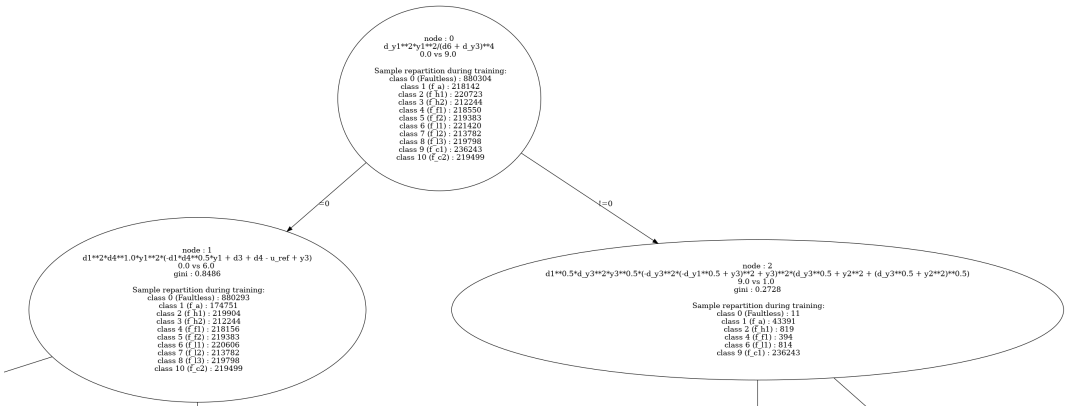


FIGURE 5.9: First Three Nodes of the Water Tanks Output Decision Tree

make predictions. The reason for this prediction time is the multivariate aspect of DT4X. It is much slower to estimate the value of a diagnosis indicator than it is to compare the value of a feature in a node for a traditional decision tree. However, the trees in a random forest are much deeper and more numerous but still manage to predict way faster than DT4X. Perhaps, DT4X can be optimized to use GPUs like the other machine learning algorithms in order to parallelize its computation process and speed up its prediction time.

The trained decision tree is too large to fit on the manuscript. Its depth is 9 and it contains 58 nodes. A small section of the tree is shown in Figure 5.9. The first thing that catches the eye is that some expressions are rather long (and some nodes have much longer expressions than what is shown here). Indeed, the expressions found by DT4X are not analytically equal to model-based ARRs. In particular, for some nodes with few samples, the expressions found tend to over-fit the data, generating these long expressions. We tried to play with the parsimony coefficient, but giving a higher one meant expressions were too short and did not pass the test **T2** of DT4X (see Section 4.2.2.1 on page 53). Despite that, the tree is really good on testing samples (99.78% accuracy and f1-score), meaning that DT4X learned to separate the classes

Hyper-parameter	Value
O	$+, -, \times, /, \sqrt{}, ^2, \|\|, sign, \geq, \leq$
ϵ	0.001
number of generations	20

TABLE 5.9: Training Hyper-Parameters of DT4X for the 3D Printer. Other hyper-parameters have default values.

correctly.

However, model-based diagnosis shows that not all classes are isolable in this system, but DT4X says that they all are. Structural analysis says that F_{f_2} and F_{l_3} are structurally non-isolable, which means that the same ARRs are sensitive to these faults. However, that does not mean that they react in the same way. In particular, here, F_{f_2} is a sensor fault that is modeled as an additive signal, whereas F_{l_3} is a leakage that is modeled as a constant parameter. They also have very different possible values. This means that it is very likely that data is affected in a different way depending on which fault occurs. Consequentially, despite being structurally non-isolable, these faults can be isolated through data.

To summarize, on this system, DT4X managed to diagnose correctly 99.78% of the unknown samples. It managed to get full diagnosability of the system and even to isolate classes that were supposed to be non-isolable. However, it did not manage to find model-based ARRs. Instead, it found data-based ARRs for the dataset used to train it, as defined in Definition 47 on page 62.

5.2.3 3D printer

Let us now test DT4X on the 3D printer defined in Section 2.3 on page 14.

5.2.3.1 Dataset

DT4X is trained with the dataset described in Section 2.3 on page 14. However, unlike in that section, sliding windows are not considered. A sample x of the dataset is made of all variables at a specific time. The split between train and test is similar to the statistical split defined in Section 2.3 on page 14, it shuffles all prints and splits in two sets. However, contrary to when working with sliding windows, a sample in the test set can not be found in the train set (neither can half of it, as with the windows).

Initially, we tried computing first order derivatives and giving that to DT4X. Then, we also tried with integrals, out of curiosity[2]. It quickly became apparent that computation is faster with integrals and accuracy is similar after the same number of nodes are trained.

Consequentially, it was decided to use integrals of the variables in the set of features. They are computed using the simpson function of scipy integrate, the Python package. Initial conditions are set to 0.

Thus, DT4X has been trained using this dataset and the hyper-parameters presented in Table 5.9. The number of generations is set low because the training time for one node with this data varies between two and six days on an AMD Ryzen 9 6900hx with 16 cores. This is due to the number of input variables and operators and to the size of the dataset. With 54 prints of around 10 minutes and a sampling rate of more than 300Hz, there is a huge number (12397858) of samples. Therefore,

[2]it was inspired by works such as D. Jung, 2020 that study performance of structural analysis based on ARR computation using integrals or derivatives

Variable Name	Variable Description
accel_x_foot	Acceleration measured by the board accelerometer in the x axis
accel_y_foot	Acceleration measured by the board accelerometer in the y axis
accel_z_foot	Acceleration measured by the board accelerometer in the z axis
accel_x_head	Acceleration measured by the nozzle accelerometer in the x axis
accel_y_head	Acceleration measured by the nozzle accelerometer in the y axis
accel_z_head	Acceleration measured by the nozzle accelerometer in the z axis
gyro_x_head	Angular velocity around the x axis measured by the gyroscope in the IMU
gyro_y_head	Angular velocity around the y axis measured by the gyroscope in the IMU
gyro_z_head	Angular velocity around the z axis measured by the gyroscope in the IMU
magn_x_head	Intensity of the magnetic field along the x axis measured by the magnetometer of the IMU
magn_y_head	Intensity of the magnetic field along the y axis measured by the magnetometer of the IMU
magn_z_head	Intensity of the magnetic field along the z axis measured by the magnetometer of the IMU
imu_temp	Temperature measured by the IMU
weight	Measured weight of the plastic thread that is equivalent to its tension
layer	Number of the current print layer
temp_bed_actual	Measured board temperature
temp_bed_target	Target board temperature
temp_tool_actual	Measured nozzle temperature
temp_tool_target	Target nozzle temperature

TABLE 5.10: Measured Variables for the 3D Printer

we decided to perform some feature selection as a preprocessing step to reduce the number of observable variables inputted in DT4X.

5.2.3.2 Preprocessing

The list of initial input variables is given in Table 5.10. To add to this list, there are integrals for each of these variables, which means 38 variables. However, not all variables bring the same amount of information with respect to diagnosis. In order to know which variables are the most informative and useful for prediction of the diagnosis classes, we ran a feature selection algorithm. We used the XGB-Classifier from XGBoost as a classification algorithm. Once trained, the attribute _feature_importance of the XGBClassifier ranks input variables in order of importance for the correct predictions. The outcome of this ranking is given in Table 5.11 on page 85. The feature importance value is a metric of how useful this variable has

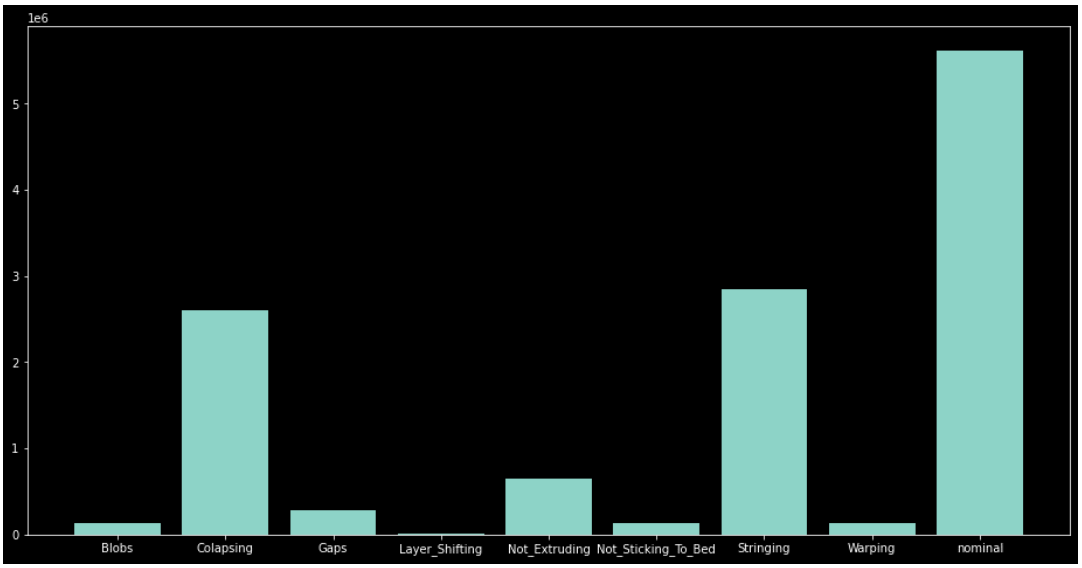


FIGURE 5.10: Sample Distribution in the Dataset

been towards the prediction. It is a proportion. The 15 first variables constitute 85% of the importance of the prediction. As a good balance between a low number of variables and a high feature importance score, they are kept for training DT4X.

It is interesting to note what those variables are. Mostly temperature measures and integrals of accelerations, which correspond to what 3D printing experts suggested (nozzle and board temperatures and printing speed). The nozzle temperature seems to be the most important factor. The layer number is also very important, this makes sense because most faults appear during the first layers of a print. Hence, just by looking at the layer number, it is possible to predict with a high likelihood whether a fault is present. Nevertheless, a very weird factor appears: the integral of the intensity of the magnetic field along the x axis. We do not have any satisfying explanation as to why.

A second preprocessing step is performed, this time to reduce the size of the dataset. Remember, this dataset is not balanced at all in terms of class size. The distribution looks like Figure 5.10. In order to both balance the distribution and reduce the overall number of samples, the practical solution is to remove samples from the largest classes. In the end, all classes contain a similar amount of samples, 8678 on average. The dataset contains 78100 samples and 15 observable variables. From this, 59325 samples constitute the training set and 19775 samples constitute the testing set. With this setup, each node of DT4X takes between one and twelve hours to train. It is still a lot, but way less than before the preprocessing.

5.2.3.3 DT4X Results

The `refit` function mentioned in Section 4.2.2.4 on page 59 is used to train the nodes one by one. A first tree is recorded when reaching depth 2. It is given in Figure 5.11 on page 86. It has an accuracy of 35.99% and its confusion matrix is given in Figure 5.12 on page 86. The fully developed tree is too large to be shown in a readable way here, but it has a depth of 7 and an accuracy of 70.02%. Its confusion matrix is given in Figure 5.13 on page 87.

Feature Importance	Variable Name
0.139814	temp_tool_target
0.100609	temp_tool_target_int
0.099177	temp_bed_target
0.071347	layer
0.065376	temp_bed_actual
0.053015	temp_tool_actual
0.050098	weight_int
0.041065	magn_x_head_int
0.034851	gyro_z_head_int
0.034337	accel_y_head_int
0.033921	imu_temp
0.032919	accel_y_foot_int
0.032515	accel_x_head_int
0.031605	accel_x_foot_int
0.027684	weight
0.023797	temp_bed_actual_int
0.022443	temp_bed_target_int
0.019976	temp_tool_actual_int
0.018627	layer_int
0.018281	accel_z_head_int
0.015477	accel_z_foot_int
0.006631	magn_y_head_int
0.005052	gyro_y_head_int
0.004331	magn_z_head_int
0.004029	gyro_x_head_int
0.003454	imu_temp_int
0.001849	accel_y_head
0.001838	magn_z_head
0.001584	accel_z_head
0.001143	magn_y_head
0.000998	gyro_y_head
0.000918	magn_x_head
0.000890	gyro_z_head
0.000245	accel_x_foot
0.000088	gyro_x_head
0.000006	accel_x_head
0.000006	accel_z_foot
0.000003	accel_y_foot

TABLE 5.11: Feature Importance of the Observable Variables for the Prediction of the 3D Printer State

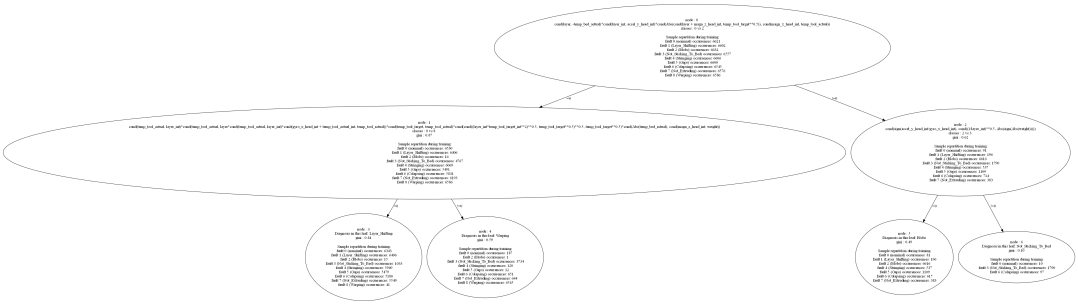


FIGURE 5.11: Decision Tree Trained by DT4X on the 3D Printer Dataset

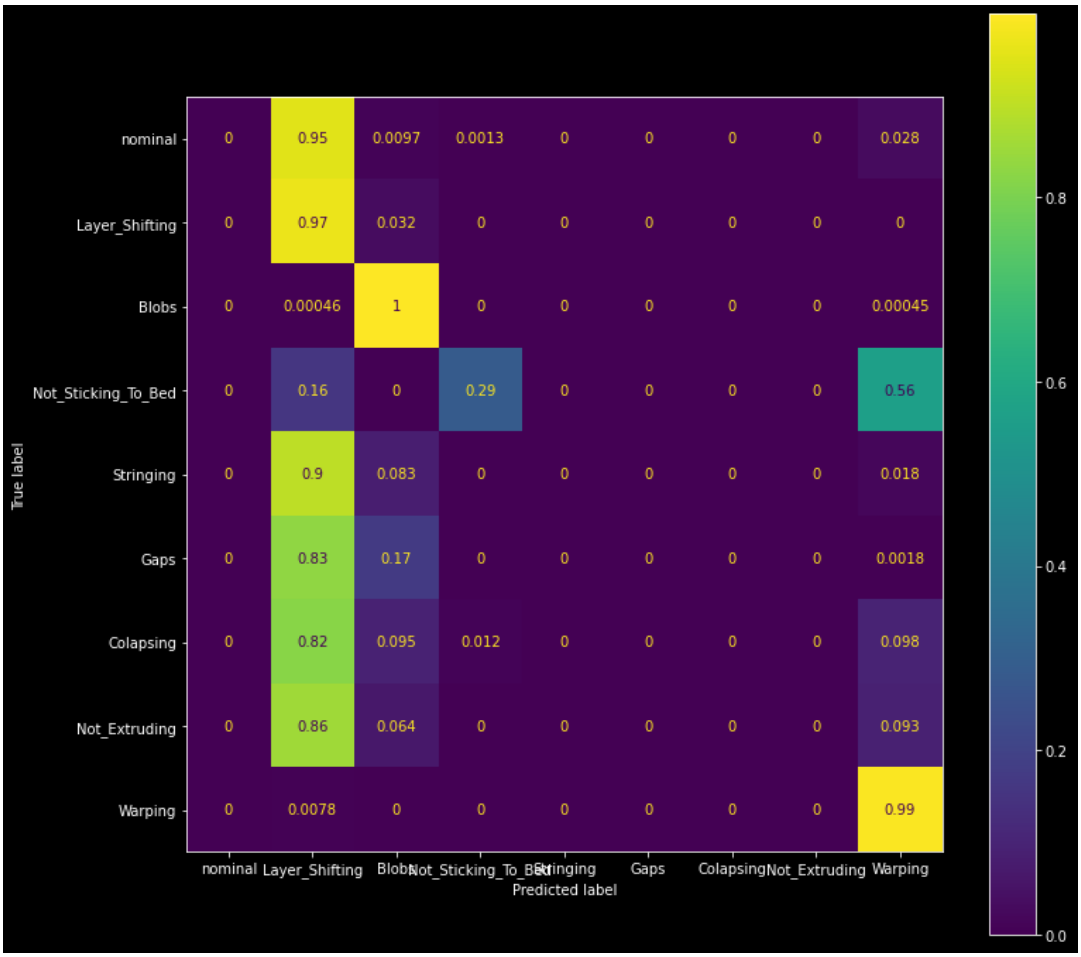


FIGURE 5.12: Confusion Matrix of the Depth 2 Decision Tree for the 3D Printer

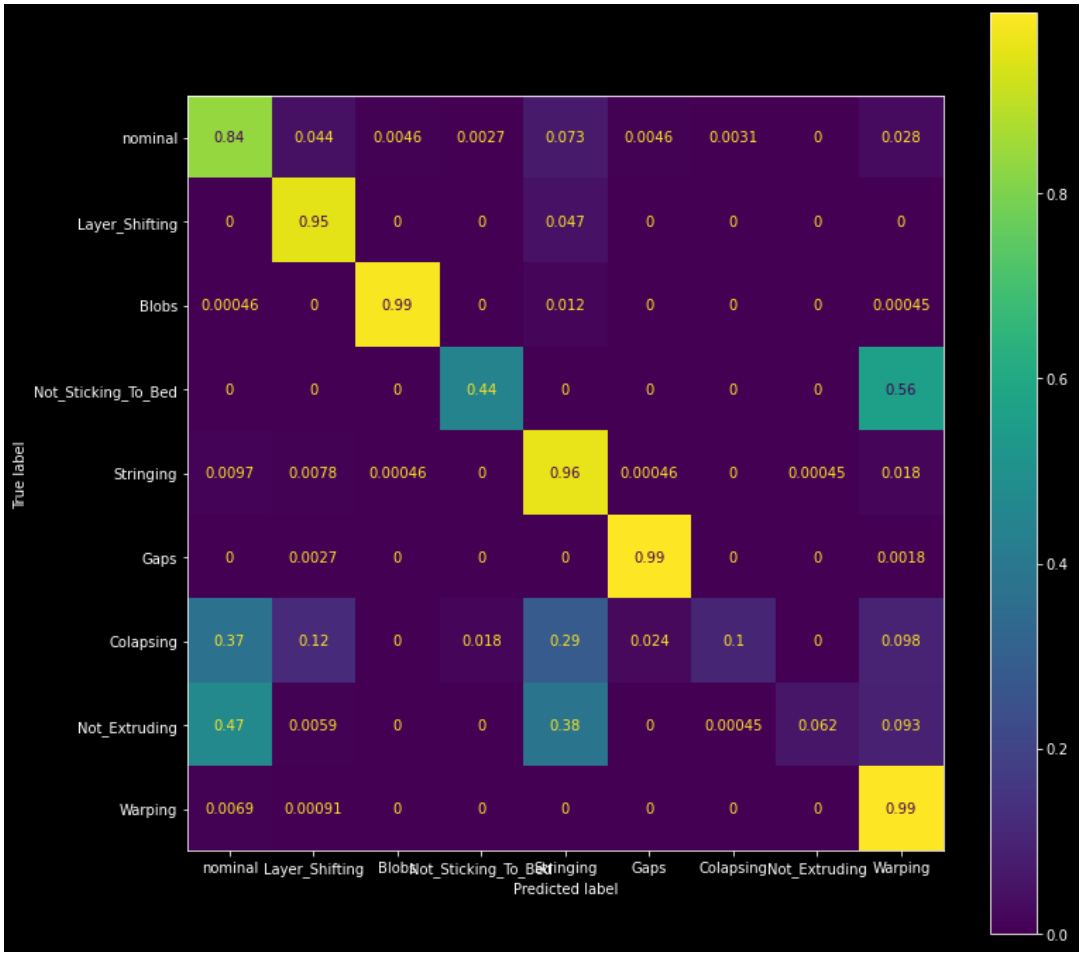


FIGURE 5.13: Confusion Matrix of the Decision Tree Trained by DT4X on the 3D Printer Dataset

5.2.3.4 Analysis

An initial purpose of the work presented in Section 2.3 on page 14 was to compare the results with those of DT4X. However, due to restrictions on the input data for explainability purposes[3], DT4X ended up tested on the same dataset, but differently preprocessed. Thus, comparing the two is not relevant anymore.

70% accuracy for the fully grown tree is far from enough if the goal is to industrialize the process. Especially since new prints can look very different from what is in the dataset making the predictions less accurate. There are multiple reasons for which DT4X is hard to adapt to this 3D printing problem. The major ones are the following:

- It is very complex to build a dataset that is representative of all printing conditions. In particular, the geometry of 3D parts can vary drastically from one print to another. Perhaps, this could be solved by building a dataset around elementary geometrical shapes from which any part can be built.
- Computation time is so high that a lot of simplifications have been made. This means that less information is available (less input variables), less operators are used than in an optimal scenario, for instance the *log* operator is not used because expert think it is not relevant for this system, but what if it is ?
- The data is very noisy. Accelerometers are very sensitive to noise and, with a high sampling frequency, noise is hard to discriminate from the actual signal.
- Labeling is not perfect, being made by hand and some time step during transitive states might be mislabeled, or data might not correspond to the system state.
- Feature selection made in the preprocessing step may be harmful to find proper diagnosis indicator analytical expressions.

The main takeaway is that, in its current state, DT4X is not fit for large-scale problems such as the 3D printer case. Perhaps, optimizing symbolic classification for use by a GPU (graphics processing unit) could speed up the process. However, it does not use matrix operations, which is what GPUs do best.

5.3 Conclusions

5.3.1 Summary

This chapter shows how DT4X is applied to four different systems and displays numerical results for the diagnosis of those systems. DT4X reaches maximum possible performance for static systems while finding data-based ARRs that are the same as the ones found through model-based diagnosis. In the context of logic circuits, DT4X is able to find logic ARRs, a new theoretical concept that we define as data-based ARRs made from logic gates. For all the static systems, DT4X reaches maximal diagnosability. A process to get rid of masked faults is used as a preprocessing step.

On dynamic systems, DT4X is faced with a major drawback, the requirement to compute noisy derivatives. It is able to match the performances of the best machine

[3]Indeed, working with features engineered from windows would not have given DT4X the opportunity to find real model-based ARRs. Had we known before conducting the machine learning tests on the printer, we would have probably done it differently. Also, in the end, DT4X did not find model-based ARRs...

learning algorithms on a clean dataset (without noise) for the water tanks example. However, the found diagnosis indicators do not seem to correspond to model-based diagnosis indicators. Actually, further analysis should be performed using formal calculus to assess whether the found data-based ARRs are equivalent to some model-based ARRs. This is considered for future works.

On the 3D printer use case, the data is noisy and not representative of all print conditions. Hence, DT4X, just as the other machine learning algorithms, struggles to reach good performances. Moreover, it faces scalability issues. Indeed, the size of the dataset and the complexity of the problem in terms of number of operators, number of observable variables and length of the found expressions make symbolic classification in DT4X slow to compute.

5.3.2 Perspectives

The proposed method to remove masked faults in the context of logic circuits can not be used as-is in continuous systems. However, masked faults plague datasets from all systems. Developing such methods when working with fault diagnosis should be more of a focus in future works.

For the water tanks use case, DT4X is able to isolate faults that are structurally non-isolable. A goal of DT4X is to find diagnosis indicators that mimic the behavior of model-based ARRs. A model-based ARR established from structural analysis and the model of the system can separate the two non-isolable faults by having different thresholds for each of them. However, structural analysis does not take into account the fact that these faults may have a different numerical impact on residuals. Interestingly enough, DT4X can inform about the cases in which the value of residuals should be considered and not only the fact that they are non-zero.

On the 3D printer case, integrals seem to be more proficient than derivatives (and easier to compute). The most likely theory is that noisy data made derivatives completely meaningless (reflecting variations in the noise much more than in the actual signal). Those questions are complex to answer, especially because of the training time of each node in the tree, that does not allow for many experiments.

Being able to test DT4X on noisy data but with coherent derivatives could potentially show that DT4X is resilient to noise. It would not solve, however, the issue that computing derivatives from noisy data is required for dynamic system diagnosis with DT4X.

Why are model-based ARRs not found by DT4X in the context of dynamic systems ? Indeed, the found expressions split the data correctly, but why did DT4X select these rather than the model-based ones that are often shorter and less complex in terms of operators ? Future works could focus on improving the loss function of symbolic classification. Indeed, DT4X uses the log loss function. This takes into account how good an expression separates the two studied classes. However, it does not take into account the other classes present in a node. In practice, most model-based ARRs are either fully sensitive to a class or not sensitive at all, meaning that all samples from a class are classified the same way. This is not a general rule, as is discussed in the next chapter with weakly detectable faults, but it is a tendency. Our idea is to use this knowledge to improve the loss function of symbolic classification by rewarding the prediction of all samples from other classes (other than the studied pair) in the same way.

Another topic that can be explored is whether the expressions found by DT4X are similar to model-based ARRs. Similar in the analytical sense, but also, similar on the set of system operating conditions, or even similar on the training dataset of DT4X.

For instance, if a diagnosis indicator of DT4X behaves exactly like a model-based ARR on the dataset, despite being different on points outside the dataset, it means that symbolic classification found a perfect solution w.r.t. the dataset and that the only way to improve the results would be to diversify the dataset to better cover the system operating conditions.

Chapter 6

Physics Informed DT4X

In this chapter, we describe how to enhance DT4X using more knowledge about the system. We consider the cases where it is possible to *a priori* obtain the structural model of the system (as defined in Section 3.1 on page 29). Is it possible to take advantage of this information and enhance DT4X with this knowledge in order to reach a higher accuracy and more relevant diagnosis indicators ? We propose PI-DT4X, a variant of DT4X that takes as input the structural model of the studied system and uses this information to enhance DT4X.

From the structural model, it is possible to assess the structural diagnosability of the system (by identifying the MSO sets). This means that DT4X loses part of its purpose. However, it still gives a lot of insight by identifying the exact diagnosis indicator equations and allowing diagnosis of unknown samples.

PI-DT4X is mainly of interest for our scientific goal, less so for the industrial goal. Still, we could imagine situations where the structural model of a system is available despite the full analytical model itself not being available.

First, we introduce background concepts, namely symbolic regression and weakly detectable faults. Then, we present PI-DT4X in depth, similarly to what was done for DT4X. Last, PI-DT4X is applied to the polybox and the water tanks use cases and conclusions are drawn from these experiments.

6.1 Background Concepts

PI-DT4X relies on symbolic regression whereas DT4X relies on symbolic classification. PI-DT4X does not use symbolic classification because of a certain type of faults: weakly detectable faults. This section aims at introducing symbolic regression and weakly detectable faults.

6.1.1 Symbolic Regression

Similarly to symbolic classification (see Section 4.1.2 on page 46), symbolic regression is a technique of machine learning and symbolic artificial intelligence where symbolic expressions or rules are used to represent relationships between input variables and an output variable. Unlike traditional statistical or machine learning approaches that rely on predefined mathematical models, symbolic regression aims to discover interpretable symbolic expressions directly from the data. Figure 6.1 on the next page illustrates the training process of symbolic regression. It consists in estimating a function $f : \mathbb{R}^n \to \mathbb{R}$, knowing pairs $(x, f(x))$ with $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ and $f(x) \in \mathbb{R}$. f is the actual function, the one that symbolic regression is trying to find the expression of, or at least approximate as best as possible. Let us call c_{best} the function found by symbolic regression that tries to be as similar to f as possible. c_{best}

FIGURE 6.1: Symbolic Regressor: during **training**, the green objects are known and the red ones are unknown and learnt.

FIGURE 6.2: Symbolic Regressor: during **testing**, the green objects are known and the red ones are unknown and predicted

can be used to find the image of a sample x by f. Figure 6.2 shows how prediction of the class of a new sample x is performed once c_{best} has been found.

Definition 52 (Symbolic Regressor)**.** *We call symbolic regressor the function c_{best} once it has been found through symbolic regression. $c_{best} : \mathbb{R}^n \to \mathbb{R}$.*

The estimation of f is done without assuming its structure; an analytical relation is therefore just discovered. There are multiple ways to carry out this estimation. As mentioned in Section 4.1.2 on page 46, a genetic algorithm can be used for that purpose. However, symbolic regression being a much more widely spread technique than symbolic classification, many algorithms have been implemented to train a symbolic regressor. Petersen et al., 2019 propose an implementation that uses recurrent neural networks to find the sequence of operators and variables that constitute f. Y. Jin et al., 2019 use bayesian methods and Udrescu and Tegmark, 2020 use physically informed neural networks. Al-Roomi and El-Hawary, 2020 present an alternative to symbolic regression that uses different mechanism, search space and building strategy while still solving the same problem. All those papers are very recent, showing a growing interest by the scientific community in the field of research that is symbolic regression. An important thing to notice is that symbolic classification can be seen as a simple extension of symbolic regression. Hence why we mentioned in Section 4.1.2 on page 46 that despite symbolic classification known implementation only using genetic algorithms, it is clear that it could be replaced with some of the algorithms mentioned here.

In the context of this manuscript, we use the `gplearn` implementation of symbolic regression that relies on a genetic algorithm (see Section 4.1.1 on page 45) and is based on the work of Poli, Langdon, and McPhee, 2008. The main reason is that it allows to save implementation time by reusing part of the code of DT4X. However, as is discussed later, other implementations could have been selected.

Symbolic regression takes as inputs a dataset $\mathcal{D} = \{(x, f(x))\}$ and a set of operators O (e.g. $+, *, -, /, \sqrt{}, ||, \log$, etc.). The genetic algorithm searches for the best expression c_{best} combining variables $(x_1, \ldots, x_n)$ and operators so that $\sum_{x \in \mathcal{D}} |c_{best}(x) - f(x)|$ is minimal (c_{best} is as close to f as possible on $\mathcal{D}$). It generates candidate solutions in the form of expressions $c : \mathbb{R}^n \to \mathbb{R}$. These expressions are represented as expression trees (Preiss, 1998, defined and illustrated in Section 4.1.2 on page 46). The candidate expressions are evaluated using a fitness function. The fitness function can vary depending on the application of symbolic regression. By default, in `gplearn`

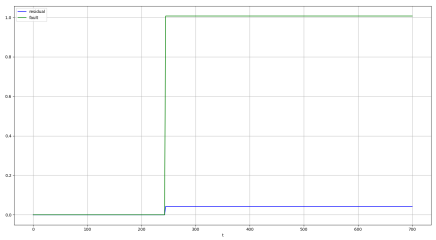


FIGURE 6.3: Detectable Fault 1

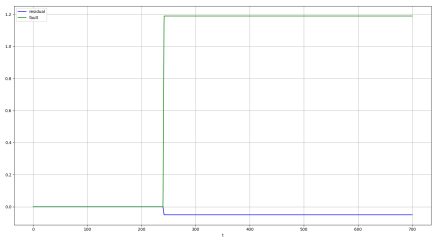


FIGURE 6.4: Detectable Fault 2

it is the mean-squared loss given by:

$$Fitness(c) = \frac{1}{|\mathcal{D}|} \sum_{(x,f(x))\in\mathcal{D}} (f(x) - c(x))^2. \tag{6.1}$$

Following the principles described in 4.1.1 on page 45, an initial population of candidate expressions is randomly generated. Then, the individuals are evaluated using the fitness function. A new generation of individuals is generated by reproducing the previous ones. Candidates with a high fitness are more likely to be used in the reproduction process. Reproduction can consist of various operations as presented in Section 4.1.2 on page 46.

This process is repeated until a stopping criterion is reached. It can be a stagnating fitness or a fitness threshold being reached by a candidate expression.

Symbolic regression has applications in various domains, including regression analysis, time series prediction, regression tasks in healthcare, finance, and engineering, as well as in scientific discovery where the goal is to identify mathematical relationships in experimental data (Iwasaki and Ishida, 2021).

6.1.2 Weakly Detectable Faults

This chapter requires defining a specific type of fault, called a weakly detectable fault (Mattias Nyberg, 2002).

Definition 53 (Weakly Detectable Fault for an MSO set)**.** *A weakly detectable fault is a type of fault that affects the behavior of the system, but only affects the behavior of the ARR associated to an MSO set for a short period of time after its occurrence, and not afterwards. It is also said to be weakly detectable for the ARR associated to the MSO set.*

It can also be seen as a fugitive fault (Soldani et al., 2006).

This type of fault is obviously very hard to detect, as it does not leave a trace in time and is not permanent. Even worse, when acquiring data from such a fault, the data after the occurrence of the fault is labeled with this fault, despite the ARR looking exactly as if the fault did not occur. It hence acts as a masked fault (see Section 5.1.2 on page 74) after some time. Figures 6.3, 6.4, 6.5 on the following page, 6.6 on the next page show different examples of detectable faults. Figure 6.7 on the following page shows an example of a non detectable fault (for this MSO set) whereas Figure 6.8 on the next page shows an example of a weakly detectable fault. They all come from the water tanks system presented in Section 3.4.1 on page 36.

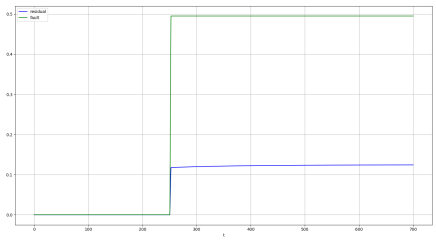


FIGURE 6.5: Detectable Fault 3

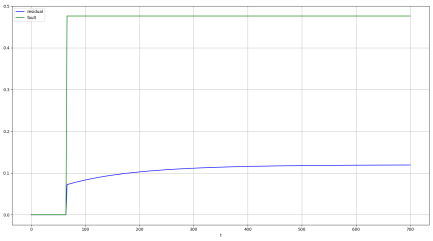


FIGURE 6.6: Detectable Fault 4

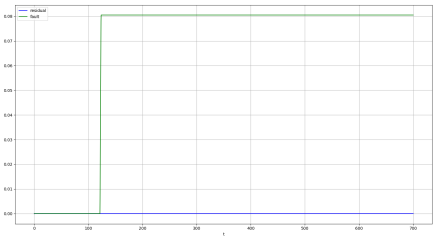


FIGURE 6.7: Not Detectable Fault

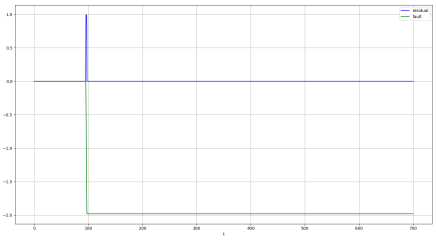


FIGURE 6.8: Weakly Detectable Fault

6.2 PI-DT4X

PI-DT4X means **P**hysics-**I**nformed **DT4X**, illustrating the fact that more system knowledge is given to DT4X.

6.2.1 PI-DT4X Principle

Similarly to DT4X, PI-DT4X trains a binary decision tree to diagnose a system. The inputs to PI-DT4X are:

- A training dataset $\mathcal{D}$ of samples (x, l) with $x \in \mathbb{R}^n$ the measurable variables of the system and $l \in C$ the associated diagnosis of the system.
- The structural model M of the system in the form of an incidence matrix (see Section 3.1 on page 29).
- A set of operators O.
- Various hyper-parameters (see Section 6.1 on page 100).

The main difference with DT4X is the addition of M. Here, prior knowledge about the system is passed both through M and O.

The core idea of PI-DT4X is to extract knowledge about model-based ARRs from the structural model and use this knowledge combined with symbolic regression to find the analytical expression of those ARRs. More precisely, from M, it is possible to identify MSO sets (see Section 3.1 on page 29). Then, in each MSO set, it is possible to know which variables are included in the associated ARR and their highest derivative order (for dynamic systems). Also, it is possible to know the fault support of this ARR. Those concepts are explained in Erik Frisk, Mattias Krysander, and Daniel Jung, 2017 and implemented in the associated Fault Diagnosis Toolbox. Up to this

point, the only difference with Section 3.4.2 on page 39 is that we also extract the highest derivative order of present variables.

However, PI-DT4X uses the same concepts as DT4X, starting with the concept of diagnosis indicator as defined in Section 4.2.1 on page 52.

In the decision tree resulting of PI-DT4X, each node $n_i \in N$ that is not a leaf contains a diagnosis indicator $\mathbf{d}_{n_i} : \mathbb{R}^n \to \mathbb{R}$ that is associated with an MSO set MSO_i. Each diagnosis indicator $\mathbf{d}_{n_i}$ is used to partition the data into two disjoint subsets, depending on whether $\mathbf{d}_{n_i}(x) = 0$ or $\mathbf{d}_{n_i}(x) \neq 0$ for all x belonging to n_i. The two subsets are then sent to a different child node. Each leaf of the resulting tree has a label that is the class predicted for the data reaching this leaf.

6.2.2 PI-DT4X Algorithm

The concept defined in Definitions 45 on page 53 and 46 on page 53 are used here as well.

Algorithm 3 gives the pseudo-code of PI-DT4X. The arrow symbol with a plus ($\leftarrow +$) means that the value is appended to the variable.

Algorithm 3 PI-DT4X pseudo-code

Inputs: $\mathcal{D}$, O, M, Untrained Decision Tree (n_0), Hyper-Parameters
Output: Trained Decision Tree with Diagnosis Indicators

```
1:  currentNodes ← n_0
2:  MSOs ← generateMSOs(M)
3:  while currentNodes is not empty do
4:      futureNodes ← emptyList
5:      for all node ∈ currentNodes do
6:          if node is pure with label then
7:              node is leaf
8:              node ← label
9:          else
10:             rank(MSOs)
11:             for all MSO ∈ MSOs do
12:                 vars ← extractVariables(MSO)
13:                 dataset_MSO ← filterData(node)
14:                 for all target ∈ vars do
15:                     c_best ← symbolicRegression(vars, target, dataset_MSO)
16:                     if check c_best then
17:                         lNode, rNode ← split according to c_best
18:                         futureNodes ← +lNode, rNode
19:                         break, break
20:                     end if
21:                 end for
22:             end for
23:             if not check c_best then
24:                 node is leaf
25:                 node ← majority label
26:             end if
27:         end if
28:     end for
29:     currentNodes ← futureNodes
30: end while
```

6.2.3 Detailed Explanation

During the training phase, a node $n_i \in N$ contains a subset of samples $\mathcal{D}_{n_i} \subset \mathcal{D}$. Each sample (x, l) in n_i verifies the conditions defined on the edges leading to n_i from the root node. At the beginning of PI-DT4X (line 1), the root node n_0 contains the entire dataset $\mathcal{D}$.

`generateMSOs` takes as input the structural model M and generates all the MSO sets for this system (line 2). This is done using the Dulmage-Mendelson decomposition (Dulmage and Mendelsohn, 1958).

PI-DT4X builds the tree starting from the root node and then going through every single node in their order of creation. The algorithm stops when no nodes are left to deal with (line 3).

When reaching a node n_i, PI-DT4X first checks whether n_i is `pure with` *label* (line 5) (see Definition 45 on page 53). If it is the case, n_i is designated as a leaf and the label *label* is associated with it (lines 6 and 7).

Otherwise, the goal is to find a new diagnosis indicator $\mathbf{d}_{n_i}$ that splits the data belonging to $\mathcal{D}_{n_i}$ (line 10 to 22).

Each non-leaf node is associated with an MSO set. When reaching node n_i, the `rank` function first eliminates all the MSO sets associated with nodes on the path towards n_i. They are not even considered for this node. Then, it ranks the remaining MSO sets according to the data present in n_i. The goal is to find the MSO set that is the most likely to split evenly the classes present in n_i, $C_{n_i} \subset C$. C_{n_i} only contains classes that are relevant in n_i (according to Definition 46 on page 53). The formula used to compute the score $score_i$ associated to each MSO set in a node n_i is the following:

$$r_i(MSO) = \frac{|\{Cl \in C_{n_i}, Cl \notin FS\}|}{|C_{n_i}|} \tag{6.2}$$

$$score_i(MSO) = \begin{cases} 2r_i(MSO) \text{ if } r_i(MSO) < 0.5 \\ 2 - 2r_i(MSO) \text{ if } r_i(MSO) > 0.5 \end{cases} \tag{6.3}$$

with FS the fault support of the ARR associated to MSO and C_{n_i} the set of classes relevant in n_i. r_i means representativity in n_i, it is a measure of how little FS covers the classes in C_{n_i}. The score of an MSO is maximal (1) if the fault support of its ARR contains exactly half the classes in C_{n_i}. It is minimal (0) either if $\forall\, Cl \in C_{n_i}, Cl \notin FS$ or if $\forall\, Cl \in C_{n_i}, Cl \in FS$. Once the scoring is done for all available MSO sets, they are ranked according to their score, with the highest score first (line 10).

Then, PI-DT4X iterates over the ranked MSO sets (line 10, Figure 6.9 on the next page). Still using the structural model, observable variables included in the ARR of the chosen MSO set are selected (line 12). Also, using the Fault Diagnosis Toolbox (Erik Frisk, Mattias Krysander, and Daniel Jung, 2017), the derivatives that are present in the ARR are also selected[1]. The selected variables are stored in *vars*.

Afterwards, $\mathcal{D}_{n_i}$ is filtered (line 13). Only data corresponding to the classes not detectable by the ARR associated to MSO_i is kept. In other words, data from classes in the fault support of the ARR is filtered out. Figure 6.10 on the facing page shows which classes are kept. Since the nominal class is never in the fault support of an ARR, it is always kept (if present in the first place). After this filtering process,

[1] In practice, all required derivatives for all MSO sets are computed beforehand, so that they can easily be used during PI-DT4X. Knowing which ones to compute is also deduced from the structural model using the toolbox. Actually, from the algorithm in the toolbox, it is only possible to know the maximum derivative order at which the variable is involved in the ARR. For instance, it says "$\ddot{x}_1$ is involved in the ARR" but using this algorithm it is not possible to know whether x_1 or $\dot{x}_1$ is also involved in the ARR. Thus, we always include all derivative orders up to the maximum.

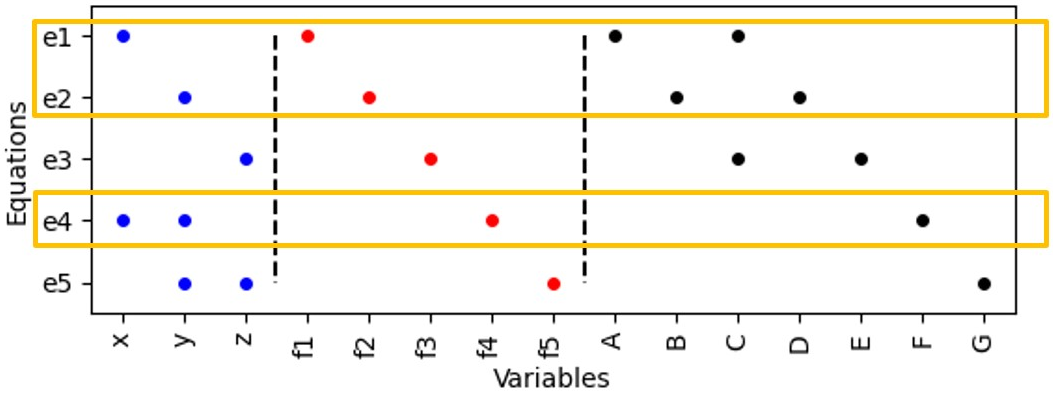


FIGURE 6.9: Selection of an MSO Set Using the Structural Model

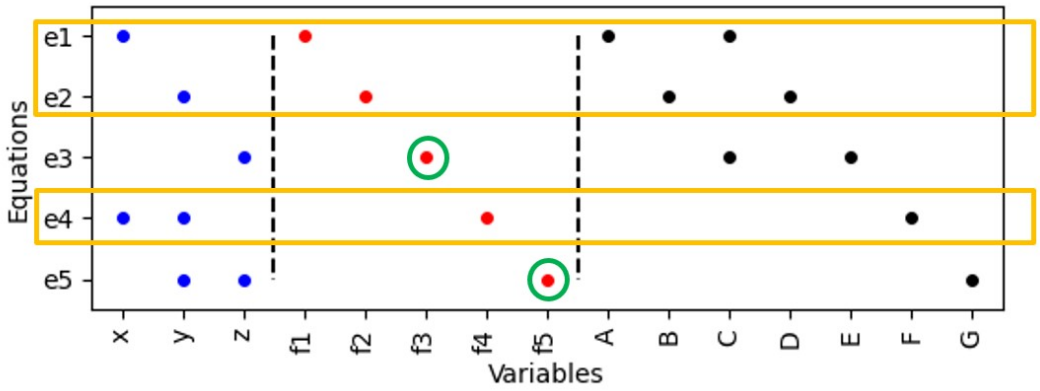


FIGURE 6.10: Classes Kept for Symbolic Regression. The green circles show fault classes that are not in the fault support of the MSO set (yellow boxes). In this case, it means that classes corresponding to fault f3 and f5 are kept. The nominal class is also kept.

the remaining dataset is called $dataset_{MSO}$. Then, PI-DT4X iterates over $vars$ (line 14). The variable selected this way is called $target$. Figure 6.11 on the next page illustrates this.

Symbolic regression is then used to find an expression made of operators from O and observable variables stored in $vars$ that fits the target variable $target$ on samples from $dataset_{MSO}$ (line 15). However, first, $target$ is removed from $vars$, otherwise fitting would be very easy and not very insightful.

Once an expression c_{best} is found by symbolic regression, it goes through three tests to make sure this expression is relevant (line 16) (see Definition 46 on page 53). Contrary to the context of DT4X, here we know that a diagnosis indicator is likely to exist. However, sometimes, symbolic regression, being based on a genetic algorithm that is intrinsically random, can stagnate and stop before finding this diagnosis indicator. The following tests are defined to ensure this is not the case:

- **T1** checks that the nominal data from the whole dataset $\mathcal{D}$ is well predicted by c_{best}. If, for at least $X_{T_1}\%$ of the samples x in the nominal data, $|c_{best}(x) - target| < \epsilon$, then the test is passed successfully. This mainly ensures that the expression is a diagnosis indicator. X_{T_1} and ϵ are hyper-parameters of PI-DT4X.

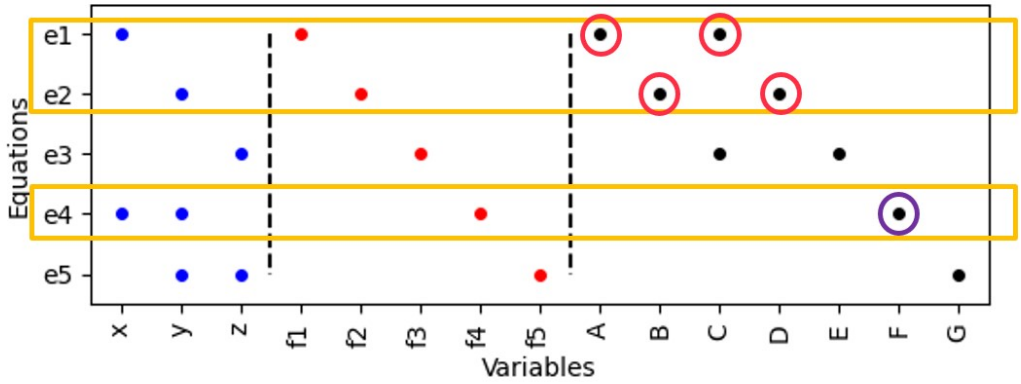


FIGURE 6.11: Variable Selection during PI-DT4X. The circled variables are in the MSO set, thus they are selected. One of them is arbitrarily set as the target.

- **T2** checks that c_{best} fits correctly $X_{T_2}\%$ of the data used to find it through symbolic regression. This test ensures that symbolic regression actually converged during the training process. X_{T_2} is a hyper-parameter of PI-DT4X.
- **T3** checks that both child nodes that would be created if c_{best} was considered valid would not be empty. Since c_{best} has been trained only on data outside the fault support, it can predict well classes for this subset but wrongly classes from its fault support. This is specifically likely when facing weakly detectable faults (see Section6.1.2 on page 93), since a weakly detectable fault would still be considered in the fault support despite the diagnosis indicator being verified for its data. Let us consider n_i whose ARR is ARR_i, with fault support FS_i; for $x \in C_{wd}$ with $C_{wd} \in FS_i$ a class corresponding to a weakly detectable fault, by definition of a weakly detectable fault, it is very likely that $|ARR_i(x)| < \epsilon$ (it is the case for all x except the ones right after the fault occurrence). Consequentially, a diagnosis indicator may not verify **T3**. The reason for which **T3** exists is not to filter out expressions that are not diagnosis indicators but rather to avoid creating empty child nodes.

If one of these tests fails, the expression is declared invalid and a different *target* is selected. If all targets lead to an invalid expression, the MSO set with the next highest score is used. If all targets in all MSO sets are tested and no valid expression is found, the node is declared a leaf and labeled with the majority class (lines 23 to 25).

However, if c_{best} verifies **T1**, **T2** and **T3**, $\mathbf{d}_{n_i} = c_{best} - target$ is declared a diagnosis indicator as it verifies all the properties of a diagnosis indicator as defined in Definition 43 on page 52 (line 16).

Therefore, the data in node n_i is `split` according to $\mathbf{d}_{n_i}$ (line 17). The algorithm evaluates $\mathbf{d}_{n_i}$ on the samples within $\mathcal{D}_{n_i}$. If the result is 0 (more or less ϵ), the sample (x, l) is sent to the left child of the current node ($lNode$). If the result is different from 0, the sample is sent to the right child ($rNode$) (line 18). Next, the two for loops are exited and the algorithm goes on to repeat the same process with the next node.

6.2.4 Design Motivations

This section aims at explaining why some choices have been made in the design of PI-DT4X.

6.2.4.1 Symbolic Regression rather than Symbolic Classification

In DT4X, symbolic classification is used to discriminate two sets A and B of samples corresponding to different classes. It can only converge if a relation exists that separates those two sets, i.e. that is (almost) zero for the samples of say A and non zero for those of B. In the context of PI-DT4X, a node is associated with an MSO set and we are targeting to find the model-based ARR corresponding to this MSO set. From structural analysis, we know which classes are in the fault support of this MSO/ARR, say those in the set B. If we use symbolic classification to discriminate the classes of the fault support of the MSO, i.e. set B, from the other classes in the set A, then we should get the correct ARR. That is without taking into account weakly detectable faults. Indeed, although a weakly detectable fault belongs to the fault support of the MSO/ARR, a large majority of samples of this class are not detectable by the ARR. Thus, symbolic classification can only fail to find a relation that discriminates samples that are actually not discriminable. This is the reason why it was decided to use symbolic regression instead of symbolic classification. Symbolic regression is fed with the data of the classes outside the fault support of the considered MSO, here those in the set A. If weakly detectable faults are in the fault support of the MSO/ARR, they are not considered and the issue is avoided.

Now, let us mention that it is more complex for symbolic regression to converge than it would have been for symbolic classification to discriminate because fitting a variable is a harder constraint than separating two sets. Also, considering less data means having less samples to train onto, which makes convergence less likely. Note that we made these two observations in the first design stages of DT4X. We experimented symbolic regression to find ARRs, and even on a case as simple as the polybox, it was not able to converge towards an ARR. In the context of PI-DT4X, giving the structural model helps narrow down the search space by giving a set of classes outside the fault support of the MSO/ARR and giving the observable variables that should be involved in the expression of the ARR. This additional "model-based" information allows symbolic regression to converge towards relevant ARRs, as will be presented in Section 6.3 on the next page.

It is worth mentioning that symbolic classification would work and probably be more efficient than symbolic regression for cases without weakly detectable faults. Also, if there was a way to identify which samples from the weakly detectable fault class(es) are misclassified by the model-based ARR, it would be more efficient to remove those samples from the training data of symbolic classification. This would be a solution in the same vein as the one applied to logic circuits in Section 5.1.2 on page 74 for which faulty samples with the same observable variable values as nominal samples were removed. This was made possible because values were binary and trivially comparable. But in the case of dynamic systems considered here, the values of observable variables are real numbers. Moreover, an ARR can evaluate to zero for two very different samples, thus there is no basis for removing undetectable faulty samples.

In the context of DT4X, weakly detectable faults are not such a prevalent issue because a node in the tree is not associated to an MSO set and to a specific ARR. If symbolic classification tries to separate two classes and one of them is or includes a weakly detectable fault for an ARR of the system, symbolic classification will just

converge towards a different data-based ARR, for which this fault is not weakly detectable.

6.2.4.2 Choice of the Target Variable

Using an arbitrary target *target* among the available observable variables *vars* is another design decision we took. Our first idea was that higher order derivatives would be the easiest targets to fit, because when solving the equations by hand, it is natural to end up with a derivative equal to a combination of the static variables (especially when coming from state-space form). However, this was experimentally disproved. Indeed, in some MSO sets, with a derivative as the target, symbolic regression did not converge, but managed to converge with a static variable as the target (despite still having the derivative in the expression). Derivatives tend to have way smaller amplitudes in the use cases we studied, perhaps this explains why they are harder to fit. However, we also noticed that when symbolic regression converges no matter which variable is the target, it does so faster (computationally) when fitting higher order derivatives. Hence, the choice to just pick targets in an arbitrary order.

6.2.5 PI-DT4X Hyper-Parameters

Hyper-parameters of PI-DT4X are very similar to the ones of DT4X. They are recapped in Table 6.1. The PI-DT4X hyper-parameters and their roles are described

PI-DT4X hyper-parameter	Default Value
purity threshold X_p	0.95
relevance threshold X_r	0.001
performance on nominal threshold X_{T_1}	0.95
indicator performance threshold X_{T_2}	0.90
ϵ	0.01
Symbolic Regression	**Default Value**
population size	5000
maximum number of generations	50
stagnation number	4
proportion of samples used	1
parsimony coefficient	0.02

TABLE 6.1: List of PI-DT4X hyper-parameters and their default values

in Section 6.2.3 on page 96.

All parameters for symbolic regression have exactly the same role and description as the ones for symbolic classification described in Section 4.2.2.3 on page 58. The main difference is that there is no classification function and ϵ is not used in symbolic regression, but ϵ from PI-DT4X actually plays the same role, as threshold for the diagnosis indicator.

6.3 Applications

6.3.1 Polybox

The polybox case study has been defined in Section 5.1.1 on page 69. In this section, we only consider the single fault scenarios, with the same dataset as in Section 5.1.1

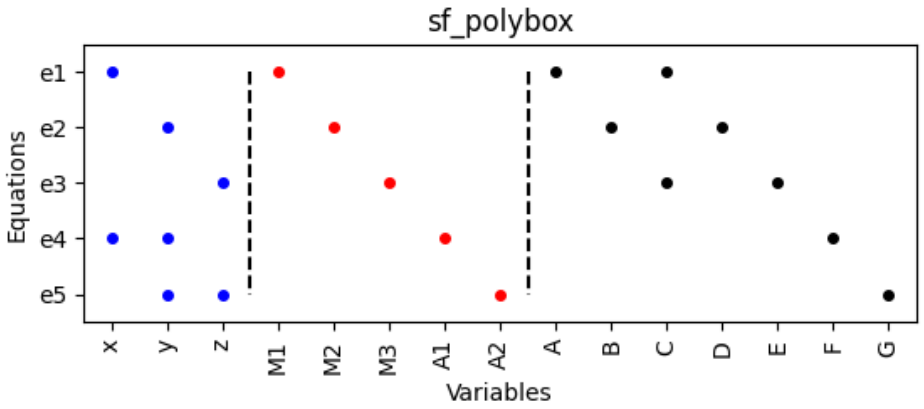


FIGURE 6.12: Structural Model of the Polybox. In the figure, M1 means f_{M1} (idem for the others) and e1 is the equation corresponding to component M1, e2 to M2, e3 to M3, e4 to A1, e5 to A2.

	Components	Observables	Fault Support
MSO_1	M2, M3, A2	B, C, D, E, G	f_{M2}, f_{M3}, f_{A2}
MSO_2	M1, M3, A1, A2	A, C, E, F, G	$f_{M1}, f_{M3}, f_{A1}, f_{A2}$
MSO_3	M1, M2, A1	A, B, C, D, F	f_{M1}, f_{M2}, f_{A1}

TABLE 6.2: List of Components and Observables in each MSO set along with the Fault Support of the Corresponding ARR. All computed from the structural model using the fault diagnosis toolbox.

on page 69. As a reminder, the structural model is given in Figure 6.12 with equations e_1, e_2 e_3 e_4 and e_5 given in Section 5.1.1.3 on page 70.

The MSO sets, as obtained through the Dulmage-Mendelson decomposition, are presented in Figure 6.13 on the following page. The fault diagnosis toolbox gives the list of observables included in the ARR of each MSO set (see Table 6.2). Since the polybox is a static system, all highest derivative orders are 0. The fault support of the associated ARR is also displayed in the table. PI-DT4X is run on the data with this information and default hyper-parameters. The output decision tree is presented in Figure 6.14 on the following page. The results are presented in Table 6.3 on the next page. They are compared with results from DT4X. It is important to keep in mind that DT4X already had maximum accuracy on the polybox case. Therefore, it makes sense that they would both have the same accuracy on the same dataset. The training times show how much faster knowing the structural model makes PI-DT4X. This is largely explained by the fact that, from the structural model knowledge, we already know that some classes are non isolable from each other. PI-DT4X identifies

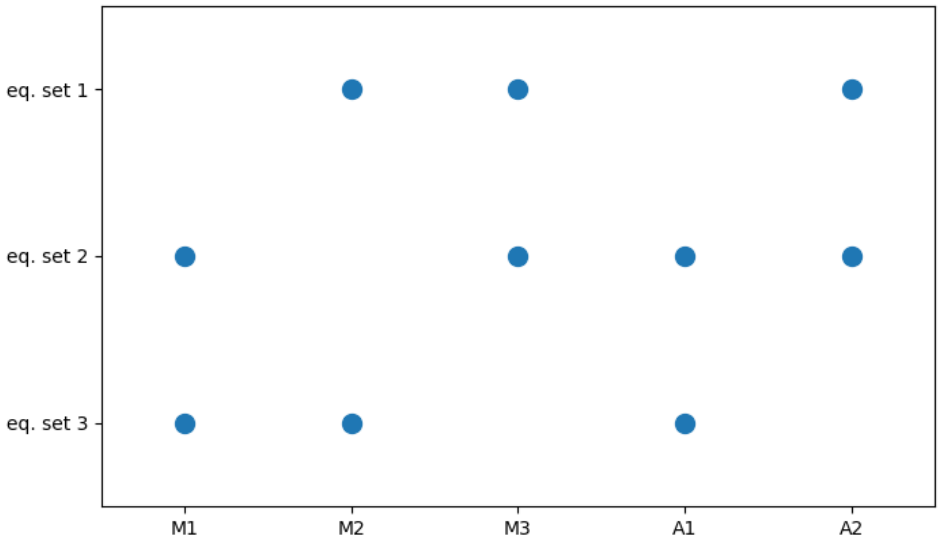


FIGURE 6.13: MSO Sets of the Polybox. A dot means that the equation of the component (horizontal axis) belongs in the equation (MSO) set (vertical axis).

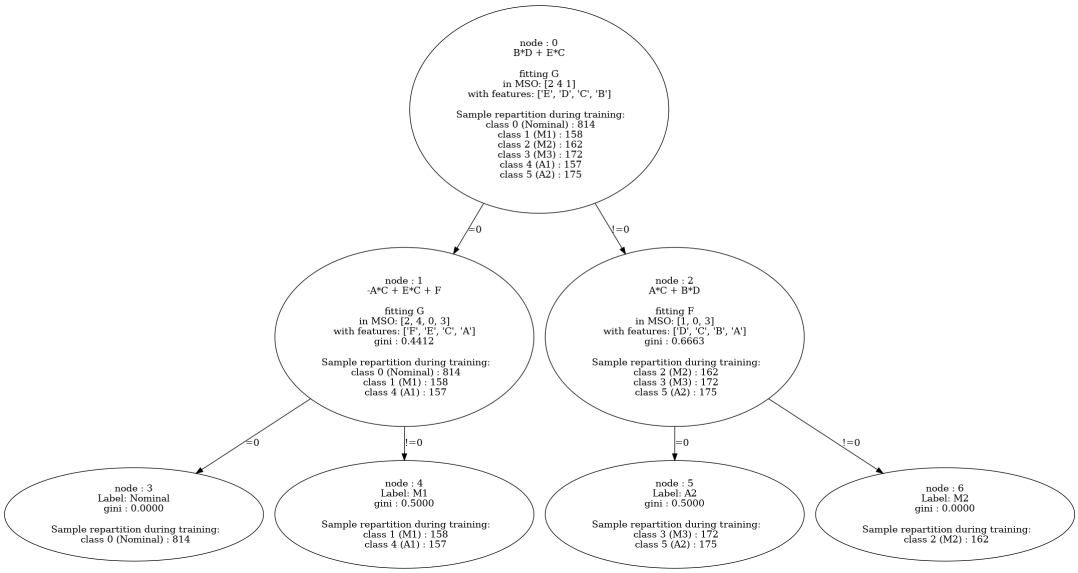


FIGURE 6.14: Decision Tree from PI-DT4X for the Single Fault Polybox

	Accuracy	f1-score	Scoring Time (s)	Training Time (s)
PI-DT4X	81.95	75.61	0.19	31.09
DT4X	81.95	75.61	0.03	1296.34

TABLE 6.3: Results from PI-DT4X compared with Results from DT4X for the polybox

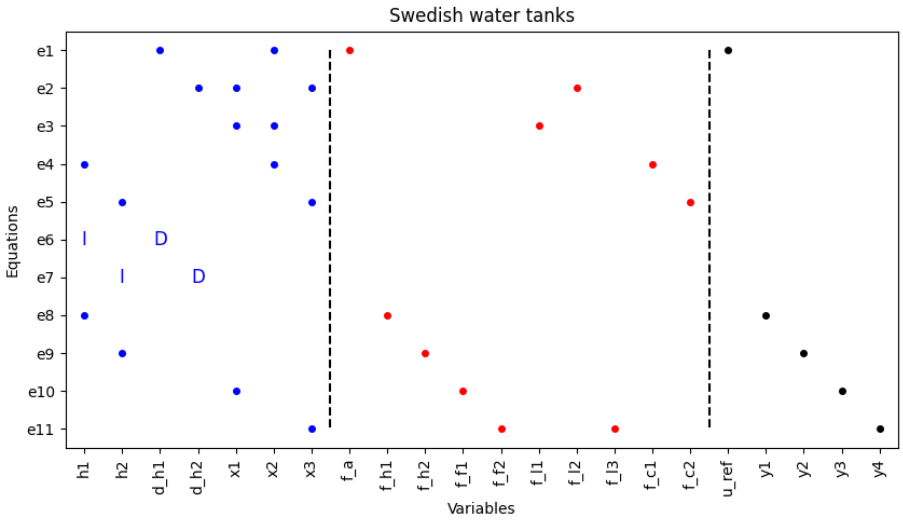


FIGURE 6.15: Structural Model of the Water Tanks. e_6 and e_7 are the differential constraints.

that no ARR can split those classes and does not try to find a diagnosis indicator that splits them.

However, DT4X has a faster prediction time. The reason is that PI-DT4X has to filter observable variables before evaluating the diagnosis indicator. It is important to note that, while it is not visible in the data presented in Table 6.3 on the facing page, PI-DT4X always finds the model-based ARRs as diagnosis indicators. It has been run more than 20 times and always finds the correct expressions right away. On the other hand, DT4X finds all the correct expressions (meaning the model-based ARRs) around fifty percent of the time. Its accuracy barely decreases when it happens. Taking that into account, PI-DT4X performs better but requires knowledge about the system that might be complex to obtain (the structural model).

Let us study how PI-DT4X fares against a more complex system in the water tanks.

6.3.2 Water Tanks

6.3.2.1 System Description

The water tanks system has been defined in Section 3.4.1 on page 36.

As a reminder, the structural model of the water tanks is presented in Figure 6.15. From the structural model, we use the fault diagnosis toolbox to compute all MSO sets and establish which observable variables are present in the ARR of each MSO set, along with their highest derivative order and the fault support of the ARR. All of this is summarized in Table 6.4 on the next page.

6.3.2.2 Dataset

The dataset used here is the same as the one for DT4X (see Section 5.2.2.1 on page 79) except that the sampling frequency is 5Hz, and second order derivatives of y_2 and

	Equations	Observables	Fault Support
MSO_1	e_5, e_9, e_{11}	y_2, y_4	$f_{h_2}, f_{f_2}, f_{l_3}, f_{c_2}$
MSO_2	e_3, e_4, e_8, e_{10}	y_1, y_3	$f_{h_1}, f_{f_1}, f_{l_1}, f_{c_1}$
MSO_3	$e_2, e_7, e_9, e_{10}, e_{11}$	$y_2, y_3, y_4, \dot{y}_2$	$f_{h_2}, f_{f_1}, f_{f_2}, f_{l_2}, f_{l_3}$
MSO_4	$e_2, e_5, e_7, e_{10}, e_{11}$	$y_3, y_4, \dot{y}_4$	$f_{f_1}, f_{f_2}, f_{l_2}, f_{l_3}, f_{c_2}$
MSO_5	$e_2, e_5, e_7, e_9, e_{10}$	$y_2, y_3, \dot{y}_2$	$f_{h_2}, f_{f_1}, f_{l_2}, f_{c_2}$
MSO_6	$e_2, e_3, e_4, e_7, e_8, e_9, e_{11}$	$y_1, y_2, y_4, \dot{y}_2$	$f_{h_1}, f_{h_2}, f_{f_2}, f_{l_1}, f_{l_2}, f_{l_3}, f_{c_1}$
MSO_7	$e_2, e_3, e_4, e_5, e_7, e_8, e_{11}$	$y_1, y_4, \dot{y}_4$	$f_{h_1}, f_{f_2}, f_{l_1}, f_{l_2}, f_{l_3}, f_{c_1}, f_{c_2}$
MSO_8	$e_2, e_3, e_4, e_5, e_7, e_8, e_9$	$y_1, y_2, \dot{y}_2$	$f_{h_1}, f_{h_2}, f_{l_1}, f_{l_2}, f_{c_1}, f_{c_2}$
MSO_9	e_1, e_4, e_6, e_8	$u_{ref}, y_1, \dot{y}_1$	f_a, f_{h_1}, f_{c_1}
MSO_{10}	$e_1, e_3, e_6, e_8, e_{10}$	$u_{ref}, y_1, y_3, \dot{y}_1$	$f_a, f_{h_1}, f_{f_1}, f_{l_1}$
MSO_{11}	$e_1, e_3, e_4, e_6, e_{10}$	$u_{ref}, y_3, \dot{y}_3$	$f_a, f_{f_1}, f_{l_1}, f_{c_1}$
MSO_{12}	$e_1, e_2, e_3, e_6, e_7, e_8, e_9, e_{11}$	$u_{ref}, y_1, y_2, y_4, \dot{y}_1, \dot{y}_2$	$f_a, f_{h_1}, f_{h_2}, f_{f_2}, f_{l_1}, f_{l_2}, f_{l_3}$
MSO_{13}	$e_1, e_2, e_3, e_5, e_6, e_7, e_8, e_{11}$	$u_{ref}, y_1, y_4, \dot{y}_1, \dot{y}_4$	$f_a, f_{h_1}, f_{f_2}, f_{l_1}, f_{l_2}, f_{l_3}, f_{c_2}$
MSO_{14}	$e_1, e_2, e_3, e_5, e_6, e_7, e_8, e_9$	$u_{ref}, y_1, y_2, \dot{y}_1, \dot{y}_2$	$f_a, f_{h_1}, f_{h_2}, f_{l_1}, f_{l_2}, f_{c_2}$
MSO_{15}	$e_1, e_2, e_3, e_4, e_6, e_7, e_9, e_{11}$	$u_{ref}, y_2, y_4, \dot{y}_2, \dot{y}_4, \ddot{y}_2$	$f_a, f_{h_2}, f_{f_2}, f_{l_1}, f_{l_2}, f_{l_3}, f_{c_1}$
MSO_{16}	$e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_{11}$	$u_{ref}, y_4, \dot{y}_4, \ddot{y}_4$	$f_a, f_{f_2}, f_{l_1}, f_{l_2}, f_{l_3}, f_{c_1}, f_{c_2}$
MSO_{17}	$e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_9$	$u_{ref}, y_2, \dot{y}_2, \ddot{y}_2$	$f_a, f_{h_2}, f_{l_1}, f_{l_2}, f_{c_1}, f_{c_2}$

TABLE 6.4: List of Equations and Observables in each MSO set along with the Fault Support of the Corresponding ARR. All computed from the structural model using the fault diagnosis toolbox. The equation numbers refer to Figure 6.15 on the previous page.

Hyper-parameter	Value
O	$+, -, \times, /, \sqrt{}, ^2$
ϵ	0.02
number of generations	150
population size	30000
parsimony coefficient	$2e^{-6}$

TABLE 6.5: Training Hyper-Parameters of PI-DT4X for the Water Tanks. Other hyper-parameters have default values.

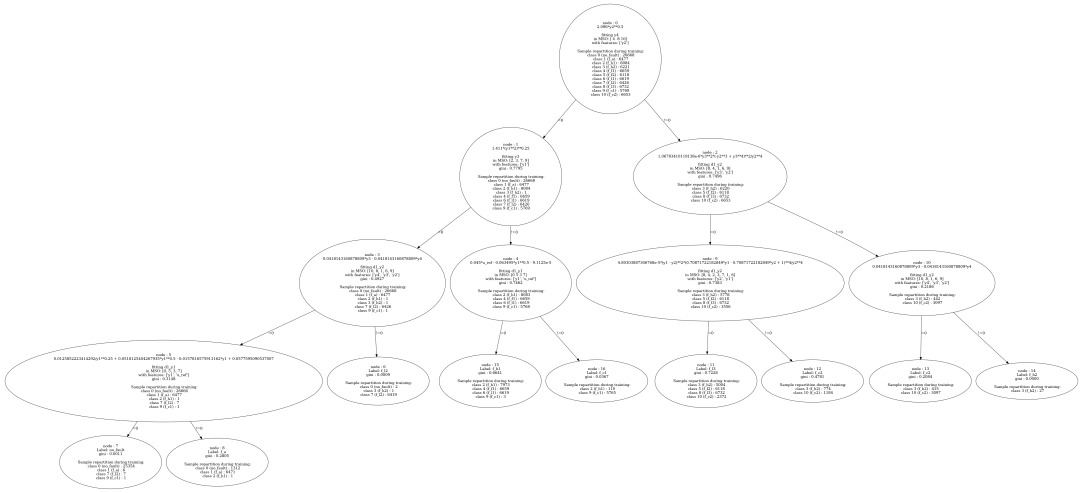


FIGURE 6.16: Decision Tree of PI-DT4X for the Water Tanks

y_4 have been computed and added as input. Then, it is split between a training and a testing dataset. The training dataset contains 308088 samples of 17 variables $(u_{ref}, y_1, y_2, y_3, y_4, \dot{y}_1, \dot{y}_2, \dot{y}_3, \dot{y}_4, \ddot{y}_2, \ddot{y}_4, d_1, d_2, d_3, d_4, d_5, d_6)$ and the corresponding label l. The testing dataset contains 77022 samples.

6.3.2.3 Results

PI-DT4X is used to train a decision tree with default hyper-parameters except the ones presented in Table 6.5. The functions are chosen according to knowledge of what type of operations the components incur. The population size and number of generations have been increased to allow the algorithm enough time to converge. ϵ and the parsimony coefficient are experimentally fine-tuned until reaching a good trade-off between exploration time and diagnosis indicator quality. This is discussed further.

The resulting diagnosis tree is presented in Figure 6.16. In addition to the information that was already present in trees from DT4X, in each non-leaf node we can find the variable that is fitted through symbolic regression, the MSO set associated with this node (in the form of a list of equations) and the features of x that are used.

From the model-based equations presented in Section 3.4.1 on page 36 we can compute a few ARRs:

$$ARR_1 = y_4 - d_6\sqrt{y_2} \tag{6.4}$$
$$ARR_2 = y_3 - d_5\sqrt{y_1} \tag{6.5}$$
$$ARR_3 = \dot{y}_2 - \frac{d_3}{d_5}y_3 + \frac{d_4}{d_6}y_4 \tag{6.6}$$
$$ARR_9 = \dot{y}_1 - d_1 u_{ref} + d_2\sqrt{y_1} \tag{6.7}$$
$$\tag{6.8}$$

In the tree, node 0 contains ARR_1, node 1 contains ARR_2, node 3 contains ARR_3 and node 4 almost contains ARR_9. Node 4 almost contains ARR_9 because there is an additional term, that is negligible compared to the ARR, at the end of the expression. This shows that PI-DT4X is able to find some model-based ARRs purely using data guided by the structural model, which is a very good sign. The nodes that contain the model-based ARRs split the data in a very convincing way, almost all data from each class end up together in the same child node.

However, when looking at the nodes that did not find model-based ARRs, a very different story is told. The classes used to train symbolic regression (classes that do not belong in the fault support of the MSO set of that node) are always classified correctly, going to the left child node. However, the other classes get split a bit randomly, which is a big concern for the final diagnosis. That is the reason why the tree has not been expanded until having only pure leaves.

The reason this is occurring is because the threshold ϵ has been relaxed. Indeed, with the default ϵ value, the test **T1** never passed, even for the real model-based ARRs, which means that the tree stopped expanding after the first two nodes (with ARRs without derivatives). Even though no noise has been added to signals, rounding errors when computing derivatives make it so that the value of the expression c_{best} found by symbolic regression is slightly off the target variable when the expression includes derivatives. In order for the correct expression to pass **T1** (or **T2** for that matter), ϵ had to be relaxed. This means that some expressions with good performance on the training data manage to pass all three tests but are not model-based ARRs and thus poorly detect samples from their fault support. This also explains that nodes 0 and 1, that use static MSO sets (in the sense that their ARR does not involve derivatives), find the correct model-based expressions right away.

A potential solution is mentioned in the perspectives, for future works to explore.

6.4 Conclusion

6.4.1 Summary

PI-DT4X is a data-driven algorithm that performs diagnosis of a system. It uses meta-knowledge from model-based diagnosis. It is an alternative to DT4X to be used when the structural model of the studied system is available (indeed, PI mean Physically Informed). Each node of the output decision tree is associated with an MSO set, computed from the structural model. Symbolic regression is used to fit an observable variable in order to learn a diagnosis indicator that corresponds to the ARR of the MSO set associated with the current node.

PI-DT4X is shown to have good performances on the polybox case and to converge much faster and avoid useless symbolic regressions using knowledge of the structural

model. On the water tanks case, it is able to find some diagnosis indicators that are model-based ARRs. However, when those model-based ARRs are not found, it is worse than DT4X at discriminating classes and diagnosing the system.

6.4.2 Perspectives

6.4.2.1 Data Normalization

In order to solve the relaxed ϵ issue, a solution would be to have a dynamic ϵ that adapts to each node, as mentioned in Chapter 4 on page 45. Another solution could perhaps be to normalize the signals in the dataset so that one well-chosen value of ϵ would fit all MSO sets. However, that would not solve the derivative computation rounding error issue.

6.4.2.2 Enhanced Symbolic Regression

The main takeaway from the state of the art of symbolic regression method is that there are many ways to perform it. They are quite recent and most of them came out during development of DT4X and PI-DT4X, hence why they have not been tested yet. Future works could focus on studying the impact of each symbolic regression algorithm. Furthermore, this notion of exploring the space of candidate solutions reminds us of another field of machine learning: reinforcement learning. This begs the question, can reinforcement learning be used as an efficient replacement to genetic algorithms ?

6.4.2.3 PSO Sets Rather than MSO Sets

Symbolic regression is really good at finding static ARRs, as shown by both the polybox results and the water tanks results, where it never fails to find a static model-based ARR. From a PSO set of equations, many residuals can be computed. Sometimes, a PSO set that contains only dynamic MSO sets (in the sense that those MSO sets all have associated ARRs that contains derivatives) can be used to compute a static ARR. It would be interesting to study if PI-DT4X could be improved by not only considering MSO sets but all the PSO sets available in each node and always choosing those that can be solved with a static ARR.

6.4.2.4 Learning the Structural Model

The main reason to use DT4X rather than PI-DT4X is that the structural model is very rarely easy to obtain. Would it be possible to learn the structural model through the data ? Our idea is to use graph neural networks and in particular graph convolutional networks (Kipf and Welling, 2016) combined with structure learning (W. Jin et al., 2020) in order to learn the structural model from the data. Indeed, as mentioned in Section 3.1 on page 29, the structural model can be represented by a bipartite graph. Links in the graph can be learned using structure learning. This, however, requires finding an objective function that makes use of the knowledge of the graph in order to learn the correct links and build the bipartite graph of the structural model.

Chapter 7

Conclusions and Perspectives

7.1 Main Contributions

This manuscript research relies on the belief that model-based diagnosis can be used in conjunction with data-driven methods to transcend their respective limitations. This thesis scientific purpose is to develop new methods that apply that principle. This thesis industrial goal is to setup these new methods on real industrial systems and analyze their performances.

First, we applied traditional machine learning to a new use case, the 3D printer, in order to study the performances and examine how to complement machine learning with model-based methods (see Chapter 2 on page 7).

After studying the literature in the domain of combined model-based and data-driven diagnosis, we proposed a new method, heavily inspired by existing methods, that combines structural analysis and machine learning in order to learn residual generators from data. This algorithm performs well but lacks explainability (see Chapter 3 on page 29). One of the goals of hybrid AI diagnosis is to use model knowledge (or model-based diagnosis meta-knowledge) to give a useful explanation as to why a fault occurs. Useful explanation is understood as able to give some information about the system's state, or give a solution to fix the fault, such as modifying the correct parameter of the system.

With the learnings from the first two experiments, we designed a novel algorithm that focuses on giving an explanation: DT4X. DT4X is a data-driven approach that trains a multivariate decision tree by finding diagnosis indicators that act as the multivariate criteria to split the data at each node. The discovered diagnosis indicators behave like analytical redundancy relations from model-based diagnosis. They are found using symbolic classification, whose goal is to try to separate two classes, the nominal class and a faulty class (see Chapter 4 on page 45).

DT4X reaches perfect performances on simulated, static systems with clean data such as the polybox and some variations. It is able to find novel analytical redundancy relations made of logic gates that we named logic ARRs when trained on logic circuit data. It reaches a good accuracy on a dynamic system with clean data but delivers complex ARRs that hardly can be considered as useful relations in terms of explainability. For a complex, very noisy, dynamic system as the 3D printer, DT4X reaches medium performances and is believed to be hardly scalable to systems of this size, at least in its current shape (see Chapter 5 on page 69).

Finally, PI-DT4X, an alternative to DT4X, takes advantage of knowing the structural model of the studied system. From the structural model, it computes MSO sets and looks for their associated ARR using symbolic regression. For that purpose, the dataset is restricted to the observable variables present in the ARR and to the faults not detectable by this ARR. Obtained trees for the polybox and the water tanks use cases are presented and show that the structural model is a powerful tool to orient the

use of data to look for model-based analytical redundancy relations (see Chapter 6 on page 91).

7.2 Conclusions

The goal we set at the beginning was to design a data-driven diagnosis method that:

- learns system information from the data;
- uses meta-knowledge (i.e. knowledge about the way the method works) from model-based methods;
- allows for expert knowledge to be exploited to enhance diagnosis;
- is explicable (i.e. allows to identify what causes a fault to occur in order to be able to correct it).

Both DT4X and PI-DT4X use meta-knowledge from model-based methods in conjunction with a symbolic method in order to find data-based ARRs.

These data-based ARRs are input-output relations that describe the behavior of a subpart of the system. ARRs are relations that, previous to this work, could only be found using the model of the system. In that sense, they also give information about the system.

Expert knowledge can and has to be used in order to choose the correct operators to give as input to both DTX4 and PI-DT4X. Additionally, for dynamic systems, expert knowledge can help find the correct derivative order to use for input data. These are the two main ways expert knowledge can be used for now. Future works could explore how to use already known relations (for instance if the equation of a component is known, meaning part of the model of the system is known) to initialize candidate solutions during symbolic algorithms, as mentioned in Section 4.3.2.1 on page 68.

Finally, SA-ML, DT4X and PI-DT4X are able to isolate the faults, which can be used to correct the system. However, one goal of explainability in our case is to provide an explanation as to why a fault occurs, not only which fault occurs. SA-ML does not provide any explanation towards that goal. DT4X and PI-DT4X both give some insights. Indeed, the path of the leaf of a fault contains multiple data-based ARRs of which we can extract a few properties: the faults that are isolated by it. Depending on the order in which the ARRs are present on the path, it might be possible to identify how components are connected, and once the fault is identified, a set of potential components causing the fault can be identified in some cases. Obviously, this is not direct explainability and requires an in-depth additional analysis.

To sum up, both DT4X and PI-DT4X are data-driven diagnosis methods that learn system information from the data by using meta-knowledge from model-based methods. They allow for some expert knowledge to be used to improve accuracy. Both provide some kind of explanation that is not explicit and requires further analysis.

7.3 Perspectives

Various perspectives are given in each chapter. We want to highlight a specific one, the automated computation of hyper-parameters of DT4X. Some hyper-parameters (namely ϵ and the parsimony coefficient) have a significant impact on the outcome of DT4X. For now, their value is chosen based on expertise and looking at the order of

magnitude of data. However, automatically estimating these parameters, and even better, adapting them along the run of DT4X would drastically ease working with DT4X and improve the results. It would provide trees of better quality in terms of computation time and found diagnosis indicators which would lead to better class separation and more accurate diagnosability. The same reasoning also works for PI-DT4X.

For DT4X, a few properties have been identified (see Section 4.2.3 on page 62). There are a lot of corollaries or theorems left to be found and the same questions can be asked about PI-DT4X. Which DT4X theorems also apply to PI-DT4X ? Is it possible to find experimental conditions that are sufficient to consider that DT4X is run in an ideal scenario (see Definition 48) ? This can be the focus of future works.

7.4 Closing Thoughts

To the best of our knowledge, DT4X is the first method to exploit symbolic classification in order to find residual generators. We took the different ideas we had around symbolic classification, trying to build an algorithm that should, theoretically, output a correct diagnosis by using the discovered residual generators, or diagnosis indicators. In hindsight, a lot of design choices were not optimal or even flat out incorrect. Exploring different algorithms for symbolic classification should have been a priority. We came to realize that many alternatives to genetic algorithms exist too late in the research process to be able to backtrack without having to re-implement many things from scratch. In the same vein, the importance of having a clean dataset when working with a real system (as opposed to simulation) was valued highly, but still not highly enough. We cannot emphasize enough how **paramount** the importance of the quality of a dataset is. The version of the 3D printer dataset presented in the manuscript is actually the third version of the dataset. We made multiple experimental designs, replaced the whole measuring system, had people from various backgrounds, including 3D printing experts, give their thoughts and still, the results show that what is learned from the dataset can not be applied to a print with a different geometry.

Despite all that, we hope that DT4X and PI-DT4X advance the state of the art about what can be achieved through symbolic classification for diagnosis. This first draft of a data-driven algorithm using meta-knowledge from model-based diagnosis can be improved in many ways, and, hopefully, future works will show how to solve the limitations we encountered during our research.

Appendix A

3D Printer Instrumentation

Two Pi cameras were used to record the print, also linked to Raspberry boards but each on a different Raspberry Pi to avoid overloading the CPU (Core Processing Unit). Indeed, it led to data loss when we ran all sensor computations, including the cameras, on the same Raspberry Pi. The first camera was placed at the same level as the bed, with a field of view covering the whole printing area (See Figure A.2 and Figure A.3).

The second one was placed directly on the nozzle, and has a focal distance of 2 cm which allows it to capture more details of the printing process and check on the filament quality when it leaves the nozzle (See Figure A.4 and Figure A.5). These signals were not used because the goal is to anticipate the apparition of faults. Cameras did not show any sign prior to fault occurrences.

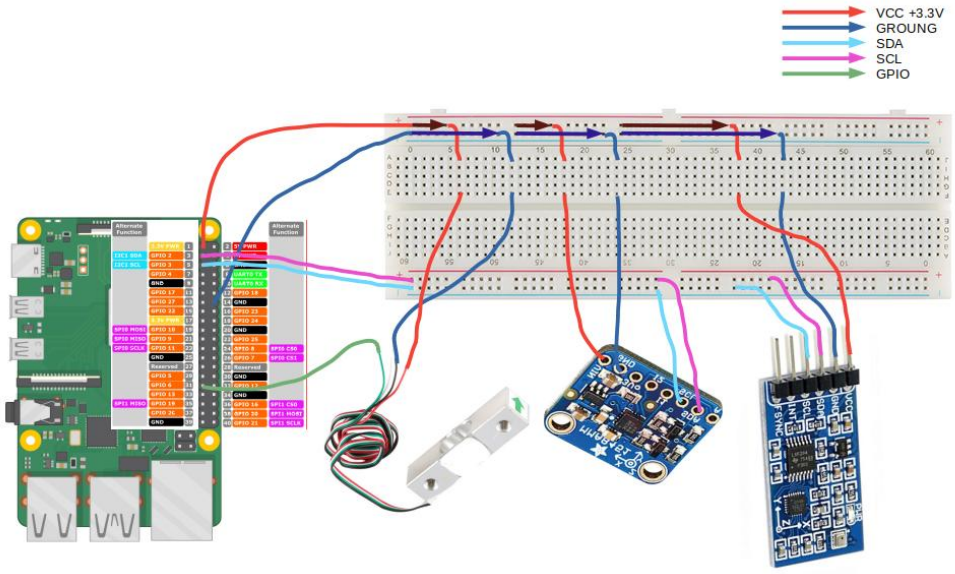


FIGURE A.1: Pin Mapping for the 3D printer Instrumentation

FIGURE A.2: Bed Camera Setup

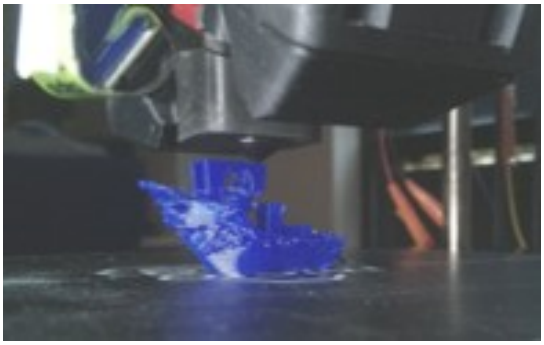


FIGURE A.3: Bed Camera View

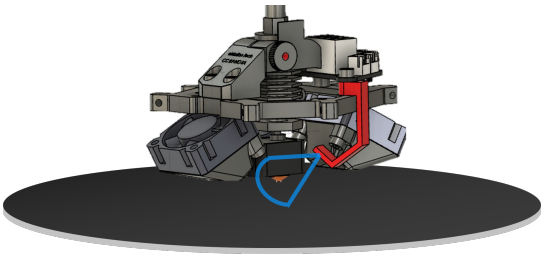


FIGURE A.4: Nozzle Camera Setup

FIGURE A.5: Nozzle Camera View

Appendix B

DT4X Applied to the Polybox

B.1 Polybox

B.1.1 Double Faults

When considering double faults, $C = \{nominal, f_{M1}, f_{M2}, f_{M3}, f_{A1}, f_{A2}, f_{M1\&M2}, f_{M1\&M3}, f_{M1\&A1}, f_{M1\&A2}, f_{M2\&M3}, f_{M2\&A1}, f_{M2\&A2}, f_{M3\&A1}, f_{M3\&A2}, f_{A1\&A2}\}$. This experiment also uses a randomly generated dataset of 7776 nominal samples and 7776 faulty samples, each being of one random fault combination (either single or double fault). A faulty component outputs the expected value plus a random modifier in $[\![-15\ ,\ 15]\!]^*$ but in the case of double faults, the value of the two modifiers are neither the same nor the opposite of each other, in order to avoid fault cancellation. This dataset is randomly split between a training set with 10887 samples and a testing set with 4665 samples. The training set is injected into DT4X with default hyperparameters and the output decision tree is shown in Figure B.1. The corresponding confusion matrix (computed on the test set) is shown in Figure B.2.

The accuracy of this decision tree compared to other default scikit-learn implementations of common machine learning algorithms are shown in Table B.1.

Algorithm	Scoring Time (s) 4665 samples	Accuracy (%)	f1 Score (%)
DT4X	0.39	64.37	58.56
sklDT	0.00	50.53	50.03
sklRF	0.08	56.03	47.61
sklLR	0.00	51.36	34.86
sklNB	0.01	51.38	36.54
sklSVM	1.04	54.98	43.14
sklKNN	0.08	55.26	46.65

TABLE B.1: Double Fault Polybox Results

B.1.2 Merged Classes Single Faults

Once the non-isolable classes have been merged, $C = \{nominal, f_{M1|A1|(M1\&A1)}, f_{M2}, f_{M3|A2|(M3\&A2)}, f_{(M1\&M2)|(M1\&M3)|(M1\&A2)|(M2\&M3)|(M2\&A1)|(M2\&A2)|(M3\&A1)|(A1\&A2)}\}$. The randomly generated dataset is composed of 7776 nominal samples and 7776 faulty samples, each being of one fault type (in C). A faulty component outputs the expected value plus a modifier in $\{-3, -2, -1, 1, 2, 3\}$. This dataset is randomly split between a training set with 12442 samples and a testing set with 3110 samples. The training set is fed to DT4X and the output decision tree is shown in Figure B.3. The corresponding confusion matrix (computed on the test set) is shown in Figure B.4.

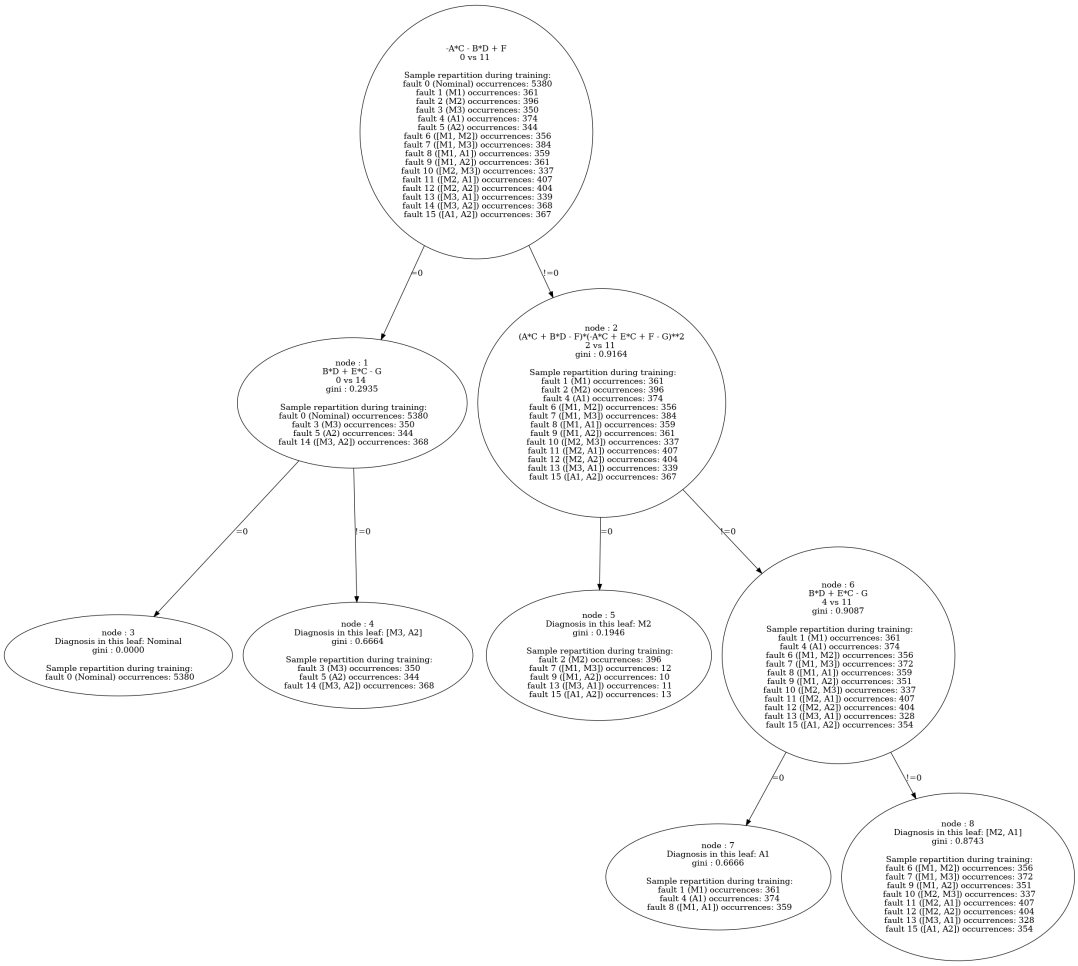


FIGURE B.1: Double Fault Polybox Decision Tree

The accuracy of this decision tree compared to other default scikit-learn implementations of common machine learning algorithms are shown in Table B.2.

B.1.3 Merged Classes Double Faults

Let us consider the double fault case when me merge the non-isolable classes together. $C = \{nominal, f_{M1|A1}, f_{M2}, f_{M3|A2}\}$. The randomly generated dataset is composed of 7776 nominal samples and 7776 faulty samples, each being of one fault type. A faulty component outputs the expected value plus a random modifier in $[\![-15\ ,\ 15]\!]^*$ but in the case of double faults, the value of the two modifiers are neither the same nor the opposite of each other, in order to avoid fault cancellation. This dataset is randomly split between a training set with 12442 samples and a testing set with 3110 samples. The training set is fed to DT4X and the output decision tree is shown in Figure B.5. The corresponding confusion matrix (computed on the test set) is shown in Figure B.6.

The accuracy of this decision tree compared to other default scikit-learn implementations of common machine learning algorithms are shown in Table B.3.

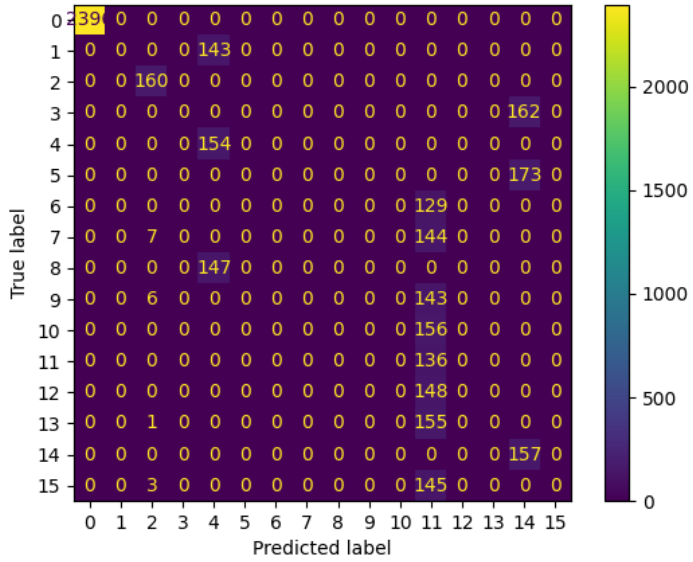


FIGURE B.2: Confusion Matrix of the Double Fault Polybox Diagnosis Tree

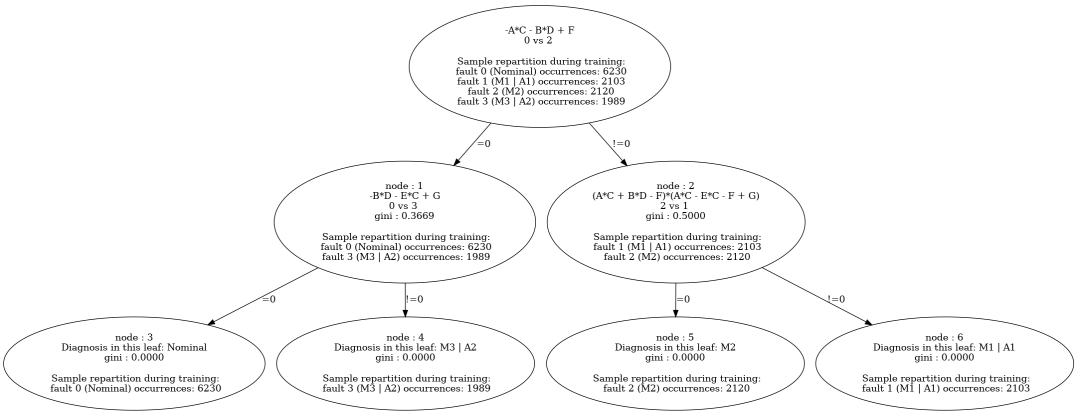


FIGURE B.3: Single Fault Polybox Decision Tree with Merged Classes

B.2 Second Polybox

B.2.1 Single Fault

A variant is presented in Figure B.7. It also has seven observable variables and if only single faults are considered, $C = \{nominal,\ f_{M1},\ f_{A1},\ f_{A2},\ f_{A3}\}$. Results for the single fault case and the double fault case are presented in Section B.2.1 and Section B.2.2 respectively of Appendix B.

$C = \{nominal,\ f_M,\ f_{A1},\ f_{A2},\ f_{A3}\}$. 7776 nominal samples and 7776 faulty samples, each being of one fault type. A faulty component outputs the expected value plus a modifier in $\{-3, -2, -1, 1, 2, 3\}$. 12442 training samples and 3110 testing samples. Decision tree: Figure B.8. Confusion matrix: Figure B.9. Comparison with other algorithms: Table B.4. Model-based equations (true residuals for comparison): Table B.5 with e_{comp} the equation that corresponds to component *comp*.

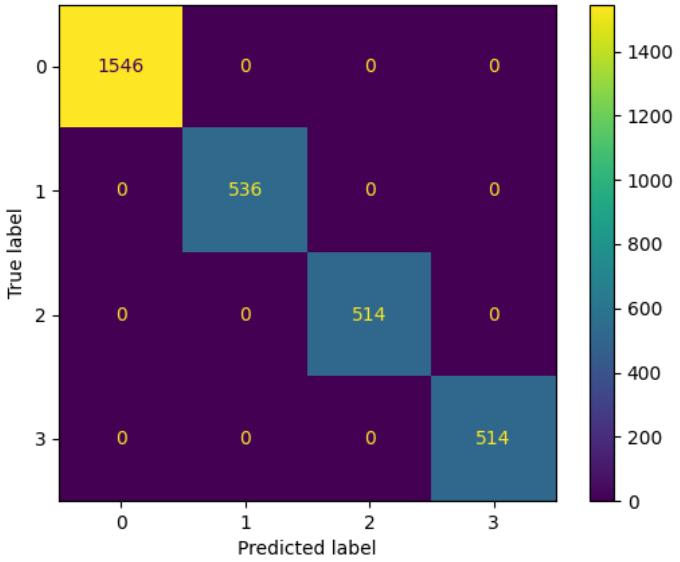


FIGURE B.4: Confusion Matrix of the Single Fault Polybox Diagnosis Tree with Merged Classes

Algorithm	Scoring Time (s) 3110 samples	Accuracy (%)	f1 Score (%)
DT4X	0.25	100.00	100.00
sklDT	0.00	60.23	59.87
sklRF	0.06	56.17	50.95
sklLR	0.00	49.71	33.01
sklNB	0.00	49.65	33.57
sklSVM	0.81	51.00	35.79
sklKNN	0.06	53.83	46.61

TABLE B.2: Single Fault Polybox with Merged Classes Results

B.2.2 Double Fault

$C = \{nominal, f_M, f_{A1}, f_{A2}, f_{A3}, f_{M\&A1}, f_{M\&A2}, f_{M\&A3}, f_{A1\&A2}, f_{A1\&A3}, f_{A2\&A3}\}$. 7776 nominal samples and 7776 faulty samples, each being of one fault type. A faulty component outputs the expected value plus a random modifier in $[\![-15\ ,\ 15]\!]^*$ but in the case of double faults, the value of the two modifiers are neither the same nor the opposite of each other, in order to avoid fault cancellation. 12442 training samples and 3110 testing samples. Decision tree: Figure B.10. Confusion matrix: Figure B.11. Comparison with other algorithms: Table B.6.

In this case, DT4X managed to always perfectly find the fault indicators of the system. They are the same as the ones obtained from model-based diagnosis.

B.3 Third Polybox

B.3.1 Single Fault

Another polybox variant is presented in Figure B.12. This variant contains an O component that outputs the opposite of its input ($out = -in$).

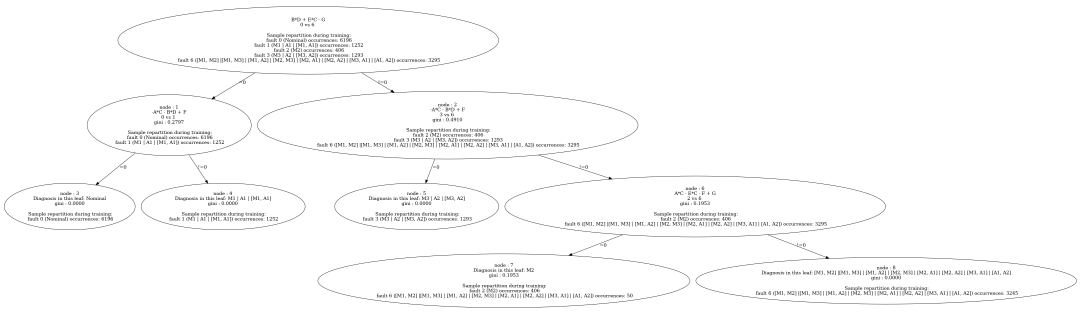


FIGURE B.5: Double Fault Polybox Decision Tree with Merged Classes

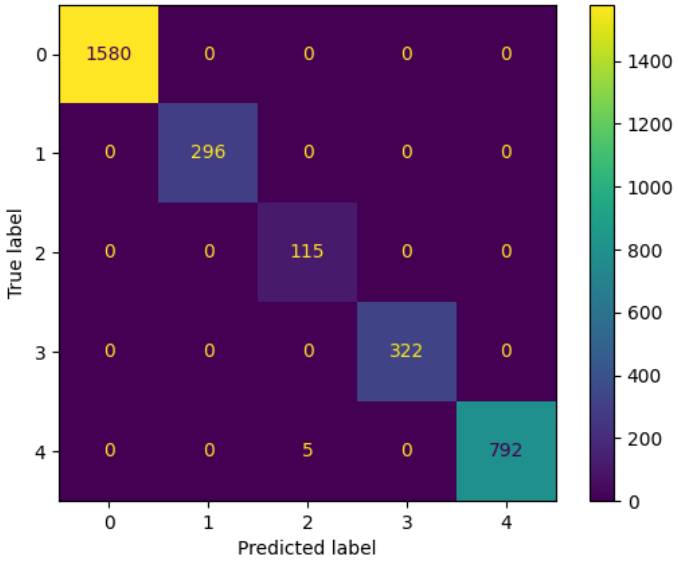


FIGURE B.6: Confusion Matrix of the Double Fault Polybox Diagnosis Tree with Merged Classes

$C = \{nominal, f_O, f_{A1}, f_{M1}, f_{M2}\}$. 2401 nominal samples and 2401 faulty samples, each being of one fault type. A faulty component outputs the expected value plus a modifier in $\{-3, -2, -1, 1, 2, 3\}$. 3842 training samples and 960 testing samples. Decision tree: Figure B.13. Confusion matrix: Figure B.14. Comparison with other algorithms: Table B.7. Model-based equations (true residuals for comparison): Table B.8.

B.3.2 Double Fault

$C = \{nominal, f_O, f_{A1}, f_{M1}, f_{M2}, f_{O\&A1}, f_{O\&M1}, f_{O\&M2}, f_{A1\&M1}, f_{A1\&M2}, f_{M1\&M2}\}$. 2401 nominal samples and 2401 faulty samples, each being of one fault type. A faulty component outputs the expected value plus a random modifier in $[\![-15\ ,\ 15]\!]^*$ but in the case of double faults, the value of the two modifiers are neither the same nor the opposite of each other, in order to avoid fault cancellation. 3842 training samples and 960 testing samples. Decision tree: Figure B.15. Confusion matrix: Figure B.16. Comparison with other algorithms: Table B.9.

This variant is a bit more tricky for DT4X because when $A = 0$ or $B = 0$ some faults are hidden (or being identical to other faults). There are ways to deal with

Algorithm	Scoring Time (s) 3110 samples	Accuracy (%)	f1 Score (%)
DT4X	0.23	99.84	99.84
sklDT	0.00	67.07	66.78
sklRF	0.06	73.44	69.04
sklLR	0.00	50.80	34.23
sklNB	0.00	54.37	44.41
sklSVM	0.69	68.26	51.06
sklKNN	0.06	66.78	62.05

TABLE B.3: Double Fault Polybox with Merged Classes Results

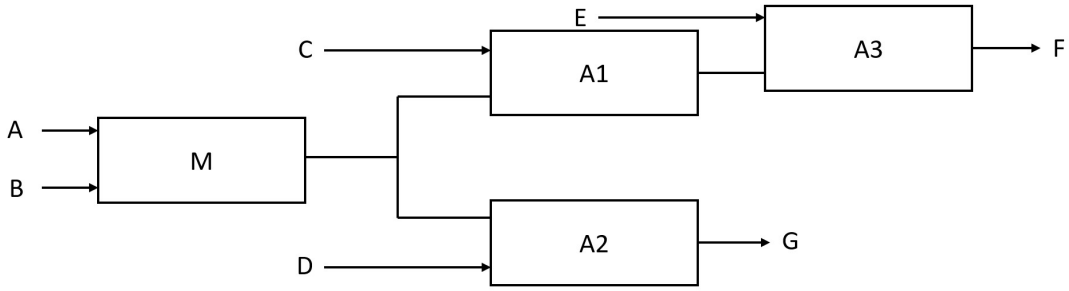


FIGURE B.7: The Second Polybox

this issue (see Section 5.1.2) but in this simple case the solution was to not input 0 at all for A or B. However, this led to issues similar to the ones encountered in the double fault cases of Section 5.1.1. Indeed, since A and B cannot be null, they do not affect the behavior of the whole expression with respect to the nominal case. Hence the expression found in the node 2 of the tree presented in Figure B.13.

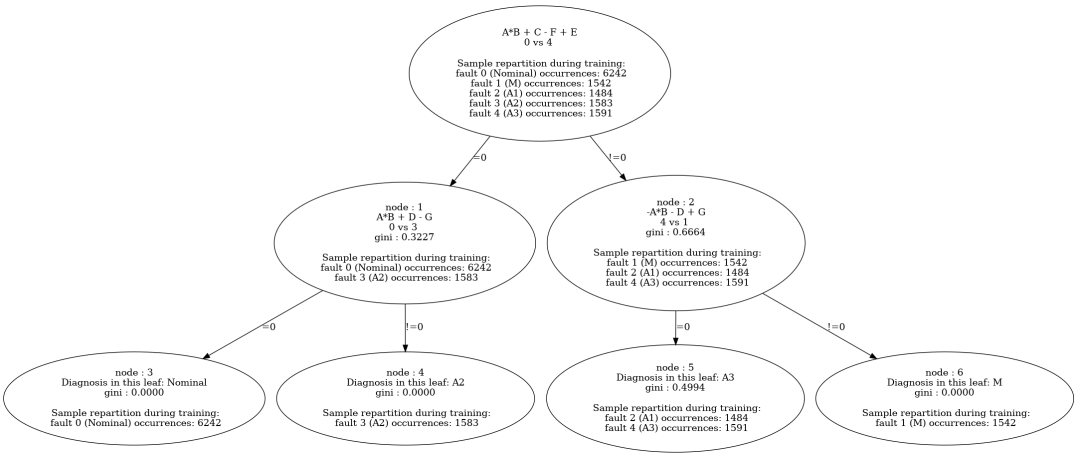


FIGURE B.8: Single Fault Second Polybox Decision Tree

Algorithm	Scoring Time (s) 3110 samples	Accuracy (%)	f1 Score (%)
DT4X	0.20	87.94	83.76
sklDT	0.00	50.35	49.81
sklRF	0.06	51.86	44.24
sklLR	0.00	49.32	32.59
sklNB	0.00	49.32	32.59
sklSVM	0.69	51.80	37.85
sklKNN	0.06	56.43	47.67

TABLE B.4: Single Fault Second Polybox Results

	Equation set	Residual Expression
MSO_1	$\{e_M, e_{A2}\}$	$A * B + D - G$
MSO_2	$\{e_M, e_{A1}, e_{A3}\}$	$A * B + C + E - F$
MSO_3	$\{e_{A1}, e_{A2}, e_{A3}\}$	$C + E - F - D + G$

TABLE B.5: Model-Based Residuals for the Second Polybox

Algorithm	Scoring Time (s) 3110 samples	Accuracy (%)	f1 Score (%)
DT4X	0.25	70.51	64.57
sklDT	0.00	57.68	57.17
sklRF	0.06	61.77	57.20
sklLR	0.00	49.55	32.83
sklNB	0.00	50.45	37.50
sklSVM	0.69	61.93	56.73
sklKNN	0.06	60.26	55.98

TABLE B.6: Double Fault Second Polybox Results

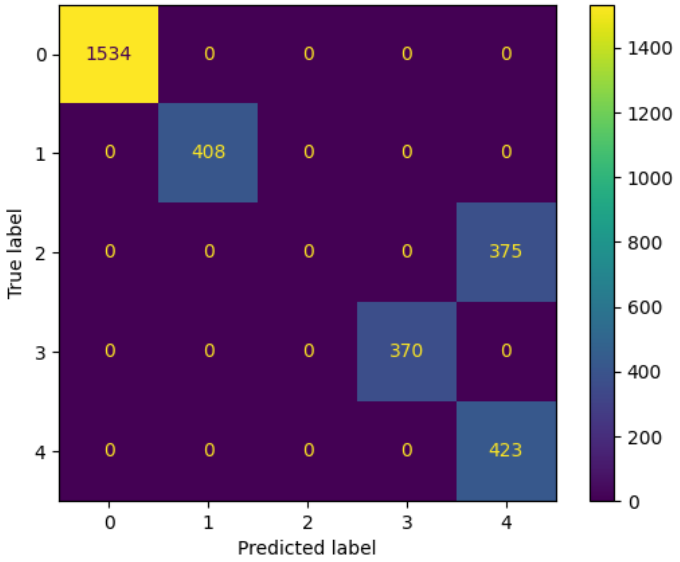


FIGURE B.9: Confusion Matrix of the Single Fault Second Polybox Diagnosis Tree

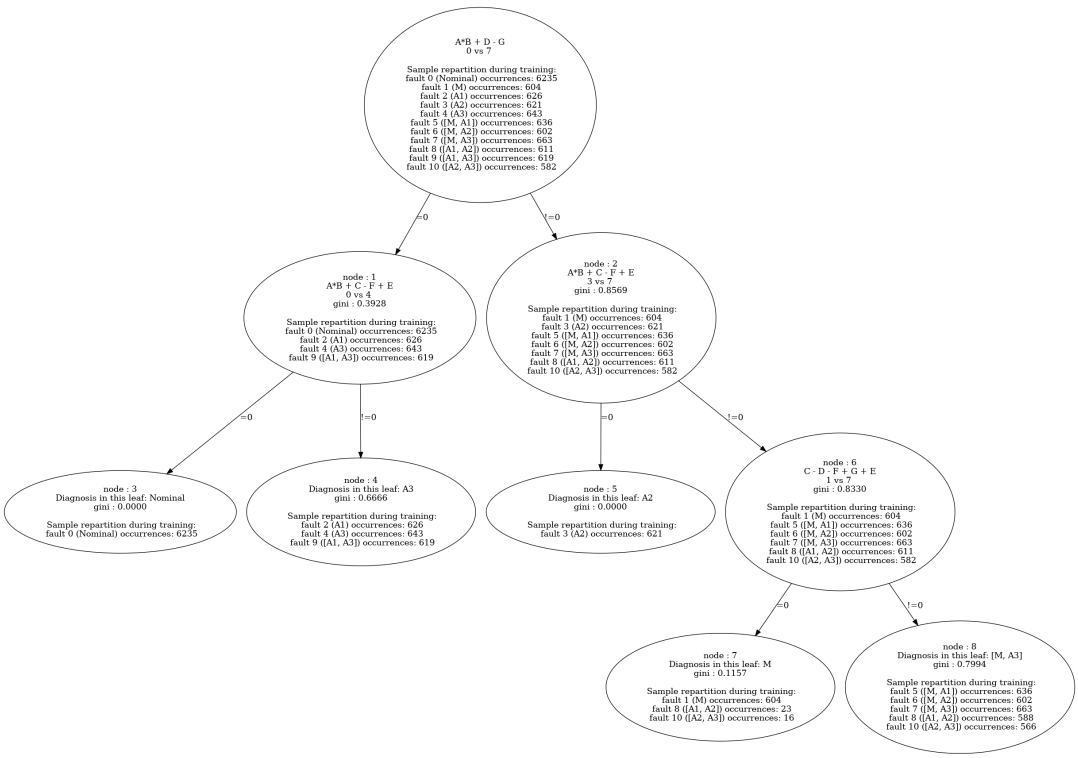


FIGURE B.10: Double Fault Second Polybox Decision Tree

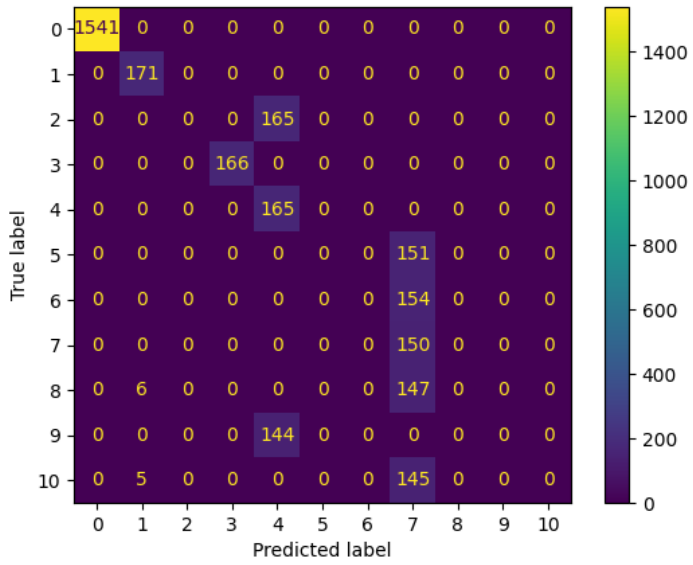


FIGURE B.11: Confusion Matrix of the Double Fault Second Polybox Diagnosis Tree

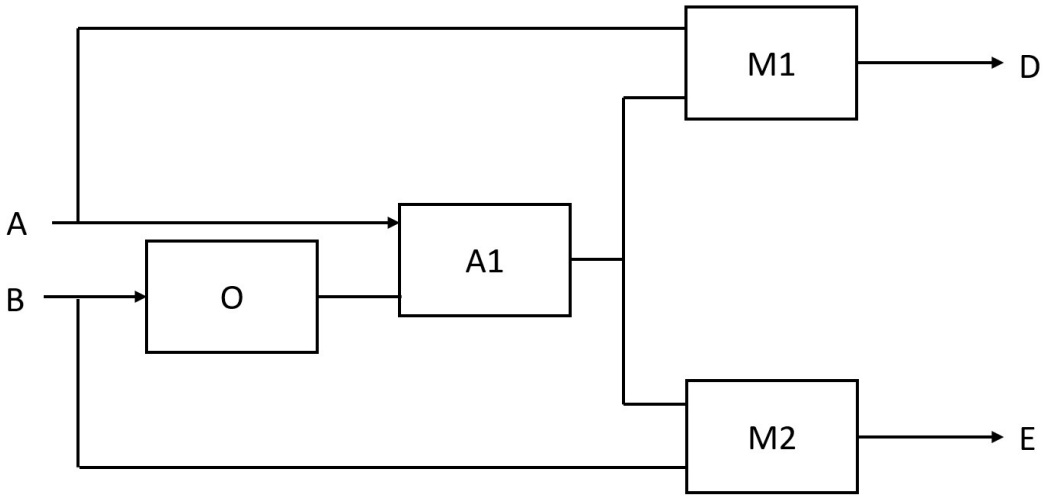


FIGURE B.12: The Third Polybox

Algorithm	Scoring Time (s) 960 samples	Accuracy (%)	f1 Score (%)
DT4X	0.07	88.65	84.58
sklDT	0.00	18.96	19.30
sklRF	0.02	22.92	20.40
sklLR	0.00	48.75	31.95
sklNB	0.00	48.75	31.95
sklSVM	0.08	48.75	31.95
sklKNN	0.01	39.90	29.53

TABLE B.7: Single Fault Third Polybox Results

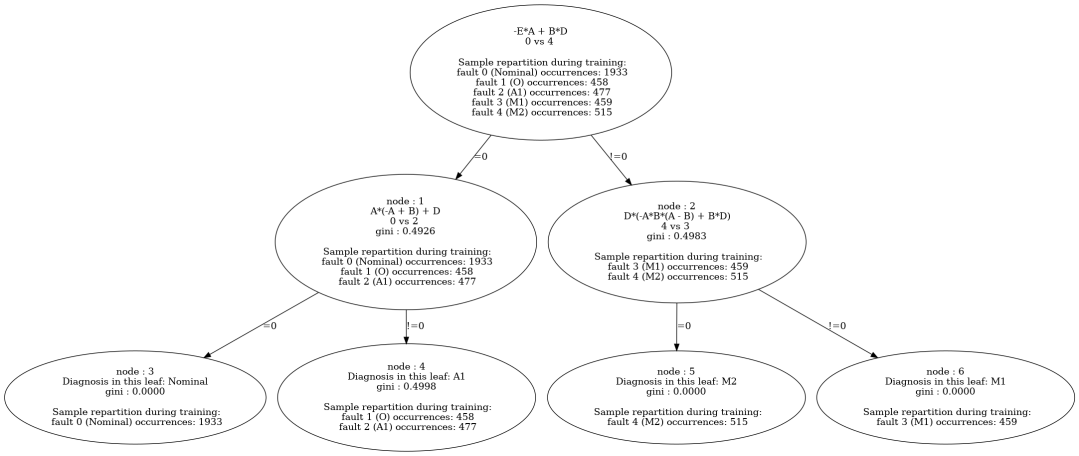


FIGURE B.13: Single Fault Third Polybox Decision Tree

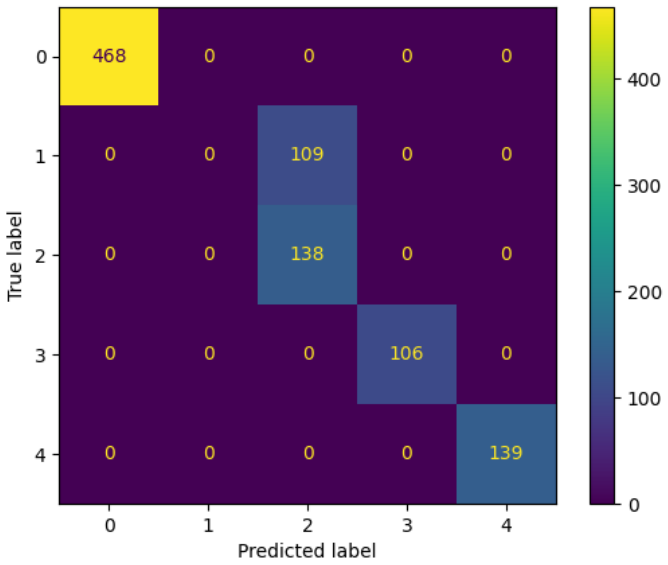


FIGURE B.14: Confusion Matrix of the Single Fault Third Polybox Diagnosis Tree

	Equation set	Residual Expression
MSO_1	$\{e_{M1}, e_{M2}\}$	$A * E - B * D$
MSO_2	$\{e_O, e_{A1}, e_{M2}\}$	$(A - B) * B - E$
MSO_3	$\{e_O, e_{A1}, e_{M1}\}$	$(A - B) * A - D$

TABLE B.8: Model-Based Residuals for the Third Polybox

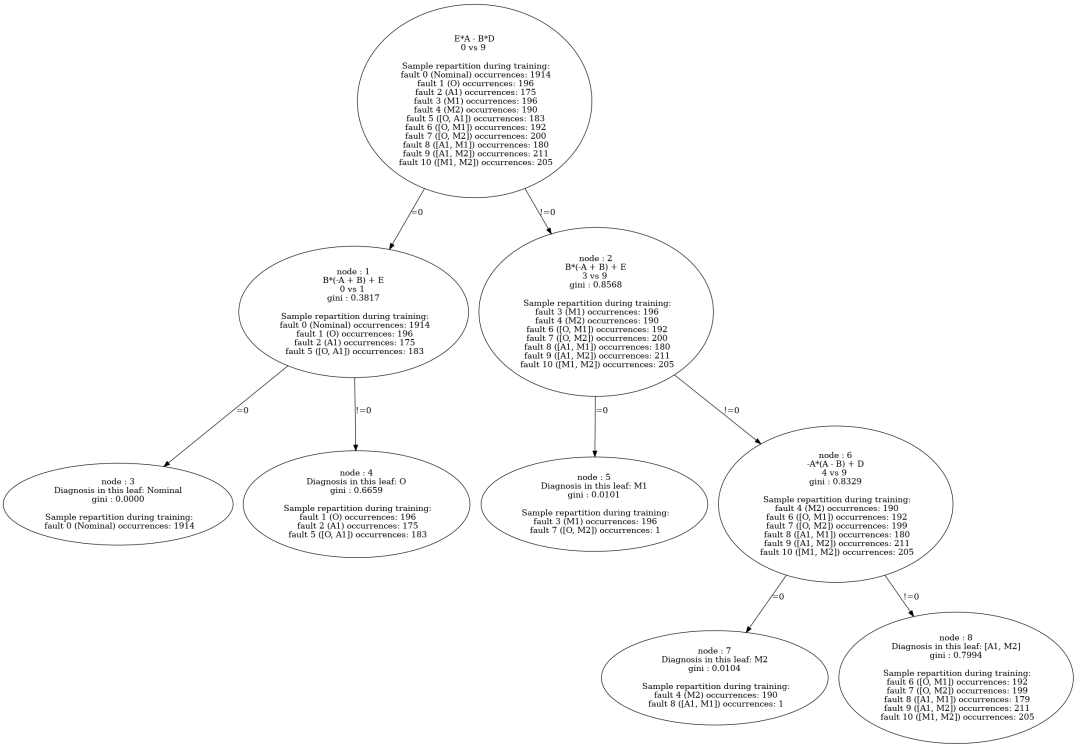


FIGURE B.15: Double Fault Third Polybox Decision Tree

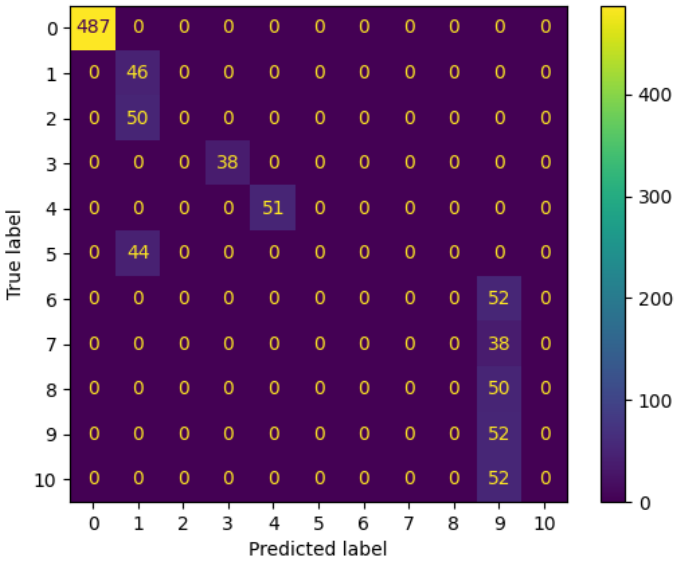


FIGURE B.16: Confusion Matrix of the Double Fault Third Polybox Diagnosis Tree

Algorithm	Scoring Time (s) 960 samples	Accuracy (%)	f1 Score (%)
DT4X	0.08	70.21	64.27
sklDT	0.00	32.40	33.26
sklRF	0.03	44.27	37.81
sklLR	0.00	50.73	34.15
sklNB	0.00	50.73	34.19
sklSVM	0.10	50.73	34.15
sklKNN	0.01	49.38	38.18

TABLE B.9: Double Fault Third Polybox Results

Bibliography

Abonyi, Janos and János Abonyi (2003). *Fuzzy model identification*. Springer.

Ademujimi, Toyosi Toriola, Michael P Brundage, and Vittaldas V Prabhu (2017). "A review of current machine learning techniques used in manufacturing diagnosis". In: *Advances in Production Management Systems. The Path to Intelligent, Collaborative and Sustainable Manufacturing: IFIP WG 5.7 International Conference, APMS 2017, Hamburg, Germany, September 3-7, 2017, Proceedings, Part I*. Springer, pp. 407–415.

Alpaydin, Ethem (2021). *Machine learning*. MIT press.

Ankenbrandt, Carol A (1991). "An extension to the theory of convergence and a proof of the time complexity of genetic algorithms". In: *Foundations of genetic algorithms*. Vol. 1. Elsevier, pp. 53–68.

Åström, Karl Johan and Peter Eykhoff (1971). "System identification—a survey". In: *Automatica* 7.2, pp. 123–162.

Bártolo, Paulo Jorge (2011). *Stereolithography: materials, processes and applications*. Springer Science & Business Media.

Basak, Jayanta and Raghu Krishnapuram (2005). "Interpretable hierarchical clustering by constructing an unsupervised decision tree". In: *IEEE transactions on knowledge and data engineering* 17.1, pp. 121–132.

Baumann, Felix and Dieter Roller (2016). "Vision based error detection for 3D printing processes". In: *MATEC web of conferences*. Vol. 59. EDP Sciences, p. 06003.

Beardwood, Jillian, J. H. Halton, and J. M. Hammersley (1959). "The shortest path through many points". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 55.4, pp. 299–327. DOI: 10.1017/S0305004100034095.

Bishop, C.M. (2006). *Pattern Recognition and Machine Learning*. p. 209. Springer.

Blanke, M. et al. (2006). *Diagnosis and Fault-Tolerant Control*. Springer-Verlag Berlin Heidelberg.

Brion, Douglas AJ and Sebastian W Pattinson (2022). "Generalisable 3D printing error detection and correction via multi-head neural networks". In: *Nature communications* 13.1, p. 4654.

Brodley, Carla E. and Paul E. Utgoff (Apr. 1995). "Multivariate decision trees". In: *Machine Learning* 19.1, pp. 45–77. ISSN: 1573-0565.

Cassar, J. and M. Staroswiecki (1997). "A structural approach for the design of failure detection and identification systems". In: *IFAC Conference on Control of Industrial Systems, vol. 30(6), pp. 841-846*.

Chen, Yi-Ming et al. (2016). "Variable-order fractional numerical differentiation for noisy signals by wavelet denoising". In: *Journal of computational physics* 311, pp. 338–347.

Chen, Z. et al. (2021). "Graph Convolutional Network-Based Method for Fault Diagnosis using a Hybrid of Measurement and Prior Knowledge." In: *IEEE transactions on cybernetics*. ISSN: 2168-2275. DOI: 10.1109/TCYB.2021.3059002.

Chow, E. Y. and A. Willsky (1984). "Analytical redundancy and the design of robust failure detection systems". In: *IEEE Transactions on Automatic Control* 29, pp. 603–614.

Console, Luca, Claudia Picardi, and D Theseider Duprè (2003). "Temporal decision trees: Model-based diagnosis of dynamic systems on-board". In: *Journal of artificial intelligence research* 19, pp. 469–512.

Costa, Vinícius G. and Carlos E. Pedreira (May 2023). "Recent advances in decision trees: an updated survey". In: *Artificial Intelligence Review* 56.5, pp. 4765–4800. ISSN: 1573-7462. DOI: 10.1007/s10462-022-10275-5.

Davidor, Yuval (1991). *Genetic Algorithms and Robotics: A heuristic strategy for optimization*. Vol. 1. World Scientific.

De Kleer, Johan and Brian C Williams (1987). "Diagnosing multiple faults". In: *Artificial intelligence* 32.1, pp. 97–130.

Deb, Kalyanmoy (1998). "Genetic algorithm in search and optimization: the technique and applications Proceedings of International Workshop on Soft Computing and Intelligent Systems". In: *ISI, Calcutta, India)* ., pp. 58–87.

Delli, Ugandhar and Shing Chang (2018). "Automated Process Monitoring in 3D Printing Using Supervised Machine Learning". In: *Procedia Manufacturing* 26, pp. 865–870.

Dolabdjian, Ch, J Fadili, and E Huertas Leyva (2002). "Classical low-pass filter and real-time wavelet-based denoising technique implemented on a DSP: a comparison study". In: *The European Physical Journal-Applied Physics* 20.2, pp. 135–140.

Duan, Zhihe et al. (2018). "Development and trend of condition monitoring and fault diagnosis of multi-sensors information fusion for rolling bearings: a review". In: *The International Journal of Advanced Manufacturing Technology* 96, pp. 803–819.

Dulmage, A. L. and N. S. Mendelsohn (1958). "Coverings of Bipartite Graphs". In: *Canadian Journal of Mathematics* 10, pp. 517–534. DOI: 10.4153/CJM-1958-052-0.

Düştegör, D. et al. (2006). "Structural Analysis of Fault Isolability in the DAMADICS benchmark". English. In: *Control Engineering Practice* 14.6, pp. 597–608. DOI: 10.1016/j.conengprac.2005.04.008.

Fioretti, Sandro and L Jetto (1994). "Low a priori statistical information model for optimal smoothing and differentiation of noisy signals". In: *International journal of adaptive control and signal processing* 8.4, pp. 305–320.

Frisk, E. and M. Nyberg (Sept. 2001). "Brief A Minimal Polynomial Basis Solution to Residual Generation for Fault Diagnosis in Linear Systems". In: *Automatica* 37.9, pp. 1417–1424. ISSN: 0005-1098. DOI: 10.1016/S0005-1098(01)00078-4.

Frisk, Erik, Mattias Krysander, and Daniel Jung (2017). "A toolbox for analysis and design of model based diagnosis systems for large scale models". In: *IFAC-PapersOnLine* 50.1, pp. 3287–3293.

Fujiwara and Toida (1982). "The complexity of fault detection problems for combinational logic circuits". In: *IEEE Transactions on computers* 100.6, pp. 555–560.

Fürnkranz, Johannes (2010). "Decision Tree". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, pp. 263–267. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_204.

García, Salvador et al. (2016). "Big data preprocessing: methods and prospects". In: *Big data analytics* 1, pp. 1–22.

Garlotta, Donald (Apr. 2001). "A Literature Review of Poly(Lactic Acid)". In: *Journal of Polymers and the Environment* 9.2, pp. 63–84.

Gerges, Firas, Germain Zouein, and Danielle Azar (2018). "Genetic Algorithms with Local Optima Handling to Solve Sudoku Puzzles". In: *Proceedings of the 2018 International Conference on Computing and Artificial Intelligence*. ICCAI '18.

Chengdu, China: Association for Computing Machinery, pp. 19–22. DOI: 10.1145/3194452.3194463.

Gilpin, Leilani H et al. (2018). "Explaining explanations: An overview of interpretability of machine learning". In: *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*. IEEE, pp. 80–89.

Goh, Guo Dong, Swee Leong Sing, and Wai Yee Yeong (2021). "A review on machine learning in 3D printing: applications, potential, and challenges". In: *Artificial Intelligence Review* 54.1, pp. 63–94.

Goupil, Louis, Elodie Chanthery, et al. (2022). "A survey on diagnosis methods combining dynamic systems structural analysis and machine learning". In: *33rd International Workshop on Principle of Diagnosis–DX 2022*.

— (2023). "Tree based diagnosis enhanced with meta knowledge". In: *34th International Workshop on Principles of Diagnosis (DX'23)*.

Goupil, Louis, Louise Travé-Massuyès, et al. (June 2024). "Tree based Diagnosis Enhanced with Meta Knowledge Applied to Dynamic Systems". In: *12th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes*. Ferrara, Italy.

Greenacre, Michael et al. (2022). "Principal component analysis". In: *Nature Reviews Methods Primers* 2.1, p. 100.

Grefenstette, John J (1993). "Genetic algorithms and machine learning". In: *Proceedings of the sixth annual conference on Computational learning theory*, pp. 3–4.

Han, Jun and Claudio Moraga (1995). "The influence of the sigmoid function parameters on the speed of backpropagation learning". In: *From Natural to Artificial Neural Computation*. Ed. by José Mira and Francisco Sandoval. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 195–201. ISBN: 978-3-540-49288-7.

He, Kun et al. (2018). "Intelligent fault diagnosis of delta 3D printers using attitude sensors based on support vector machines". In: *Sensors* 18.4, p. 1298.

Holland, John H. (1992). "Genetic Algorithms". In: *Scientific American* 267.1, pp. 66–73. ISSN: 00368733, 19467087. URL: http://www.jstor.org/stable/24939139 (visited on 04/12/2024).

Hollerbach, John, Wisama Khalil, and Maxime Gautier (2016). "Model identification". In: *Springer handbook of robotics*, pp. 113–138.

Iwasaki, Yuma and Masahiko Ishida (2021). "Data-driven formulation of natural laws by recursive-LASSO-based symbolic regression". In: *arXiv*.

Izenman, Alan Julian (2013). "Linear discriminant analysis". In: *Modern multivariate statistical techniques*. Springer, pp. 237–280.

Jahanirad, H (2019). "Efficient reliability evaluation of combinational and sequential logic circuits". In: *Journal of Computational Electronics* 18.1, pp. 343–355.

Jain, Jawahar et al. (1997). "A survey of techniques for formal verification of combinational circuits". In: *Proceedings International Conference on Computer Design VLSI in Computers and Processors*. IEEE, pp. 445–454.

Jin, Wei et al. (2020). "Graph structure learning for robust graph neural networks". In: *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 66–74.

Jin, Ying et al. (2019). "Bayesian symbolic regression". In: *arXiv*.

Jin, Z, Z Zhang, and GX Gu (2019). *Autonomous in-situ correction of fused deposition modeling printers using computer vision and deep learning. Manuf Lett 22: 11–15.*

Joyce, James (2003). "Bayes' theorem". In.

Jung, D. (2020). "Residual Generation using Physically-Based Grey-Box Recurrent Neural Networks for Engine Fault Diagnosis". In: *arXiv preprint arXiv:2008.04644*. URL: https://arxiv.org/abs/2008.04644.

Jung, D. and C. Sundstrom (2017). "A Combined Data-Driven and Model-Based Residual Selection Algorithm for Fault Detection and Isolation". In: *IEEE Transactions on Control Systems Technology* 27.2, pp. 616–630. DOI: 10.1109/tcst.2017.2773514.

Keesman, Karel J (2011). *System identification: an introduction*. Springer Science & Business Media.

Kipf, Thomas N and Max Welling (2016). "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907*.

Kleinbaum, David G et al. (2002). *Logistic regression*. Springer.

Kotsiantis, Sotiris B (2013). "Decision trees: a recent overview". In: *Artificial Intelligence Review* 39, pp. 261–283.

Kotsiantis, Sotiris B, Ioannis Zaharakis, P Pintelas, et al. (2007). "Supervised machine learning: A review of classification techniques". In: *Emerging artificial intelligence applications in computer engineering* 160.1, pp. 3–24.

Krüger, Jacob et al. (2018). "Towards a Better Understanding of Software Features and Their Characteristics: A Case Study of Marlin". In: *Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems*. Association for Computing Machinery, pp. 105–112.

Krysander, M., J. Åslund, and E. Frisk (Jan. 2010). "A Structural Algorithm for Finding Testable Sub-models and Multiple Fault Isolability Analysis". In: *21st Annual Workshop Proceedings, phm society*.

Lepot, Mathieu, Jean-Baptiste Aubin, and François H.L.R. Clemens (2017). "Interpolation in Time Series: An Introductive Overview of Existing Methods, Their Performance Criteria and Uncertainty Assessment". In: *Water* 9.10.

Li, Kang, Jian-Xun Peng, and George W Irwin (2005). "A fast nonlinear model identification method". In: *IEEE Transactions on Automatic Control* 50.8, pp. 1211–1216.

Li, Shi et al. (2020). "An adaptive data fusion strategy for fault diagnosis based on the convolutional neural network". In: *Measurement* 165, p. 108122.

Lissovoi, Andrei and Pietro S Oliveto (2019). "On the time and space complexity of genetic programming for evolving Boolean conjunctions". In: *Journal of Artificial Intelligence Research* 66, pp. 655–689.

Liu, Siyu et al. (2022). "Simple Structural Descriptor Obtained from Symbolic Classification for Predicting the Oxygen Vacancy Defect Formation of Perovskites". In: *ACS Applied Materials & Interfaces* 14.9. PMID: 35196010, pp. 11758–11767. DOI: 10.1021/acsami.1c24003.

Ljung, Lennart (1995). *System identification*. Univ.

Loja, Rene Vinicio Sanchez et al. (2020). "One-shot Fault Diagnosis of 3D Printers Through Improved Feature Space Learning". In: *Ieee Transactions on Industrial Electronics* 2020.2020.

Lu, Shyue-Kung et al. (2003). "Combinational circuit fault diagnosis using logic emulation". In: *2003 IEEE International Symposium on Circuits and Systems (ISCAS)*. Vol. 5. IEEE, pp. V–V.

Manning, Timmy, Roy D Sleator, and Paul Walsh (2013). "Naturally selecting solutions: the use of genetic algorithms in bioinformatics". In: *Bioengineered* 4.5, pp. 266–278.

Mohammadi, Arman et al. (2023). *Analysis of Numerical Integration in RNN-Based Residuals for Fault Diagnosis of Dynamic Systems*. arXiv: 2305.04670 [cs.LG].

Mothilal, Ramaravind K, Amit Sharma, and Chenhao Tan (2020). "Explaining machine learning classifiers through diverse counterfactual explanations". In: *Proceedings of the 2020 conference on fairness, accountability, and transparency*, pp. 607–617.

Murota, K. (Jan. 2009). *Matrices and Matroids for Systems Analysis*. Vol. 20. Springer. ISBN: 978-3-642-03993-5. DOI: 10.1007/978-3-642-03994-2.

Nyberg, Mattias (2002). "Criterions for detectability and strong detectability of faults in linear systems". In: *International Journal of Control* 75.7, pp. 490–501.

Oliveto, Pietro S and Carsten Witt (2015). "Improved time complexity analysis of the simple genetic algorithm". In: *Theoretical Computer Science* 605, pp. 21–41.

Pérez, G. et al. (July 2017). "Fault-driven structural diagnosis approach in a distributed context". In: *20th World Congress of the International Federation of Automatic Control*, pp.14819–14824. URL: https://hal.archives-ouvertes.fr/hal-01579467.

Pérez-Zuñiga, CG et al. (2018). "Decentralized diagnosis via structural analysis and integer programming". In: *IFAC-PapersOnLine* 51.24, pp. 168–175.

Petersen, Brenden K et al. (2019). "Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients". In: *arXiv preprint arXiv:1912.04871*.

Peterson, Leif E (2009). "K-nearest neighbor". In: *Scholarpedia* 4.2, p. 1883.

Petsiuk, Aliaksei L. and Joshua M. Pearce (2020). "Open source computer vision-based layer-wise 3D printing analysis". In: *Additive Manufacturing* 36, p. 101473.

Poli, Riccardo, William B. Langdon, and Nicholas Freitag McPhee (2008). *A field guide to genetic programming*. (With contributions by J. R. Koza). Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`.

Preiss, Bruno R. (1998). "Expression Trees". In: *Retrieved December* 20, p. 2010.

Priyam, Anuja et al. (2013). "Comparative analysis of decision tree classification algorithms". In: *International Journal of current engineering and technology* 3.2, pp. 334–337.

Refaeilzadeh, Payam, Lei Tang, and Huan Liu (2009). "Cross-Validation". In: *Encyclopedia of Database Systems*. Springer US, pp. 532–538.

Riaz, Muhammad, Nasir Abbas, and Ronald JMM Does (2011). "Improving the performance of CUSUM charts". In: *Quality and Reliability Engineering International* 27.4, pp. 415–424.

Riedmiller, Martin and A Lernen (2014). "Multi layer perceptron". In: *Machine Learning Lab Special Lecture, University of Freiburg* 24.

Rigatti, Steven J (2017). "Random forest". In: *Journal of Insurance Medicine* 47.1, pp. 31–39.

Rojas, Raul and Raúl Rojas (1996). "The backpropagation algorithm". In: *Neural networks: a systematic introduction*, pp. 149–182.

Rong, Shen and Zhang Bao-Wen (2018). "The research of regression model in machine learning field". In: *MATEC Web of Conferences*. Vol. 176. EDP Sciences, p. 01033.

Al-Roomi, Ali R and Mohamed E El-Hawary (2020). "Universal functions originator". In: *Applied Soft Computing* 94, p. 106417.

Rudin, Cynthia (2019). "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead". In: *Nature machine intelligence* 1.5, pp. 206–215.

Shao, Haidong et al. (2021). "A novel approach of multisensory fusion to collaborative fault diagnosis in maintenance". In: *Information Fusion* 74, pp. 65–76.

Singh, Amanpreet, Narina Thakur, and Aakanksha Sharma (2016). "A review of supervised machine learning algorithms". In: *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 1310–1315.

Slimani, Amel et al. (2018). "Fusion of model-based and data-based fault diagnosis approaches". In: *IFAC-PapersOnLine* 51.24, pp. 1205–1211.

Šljivic, M et al. (Oct. 2019). "Comparing the accuracy of 3D slicer software in printed enduse parts". In: *IOP Conference Series: Materials Science and Engineering* 659.1, p. 012082.

Smith, Alexander et al. (2005). "Fault diagnosis and logic debugging using Boolean satisfiability". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24.10, pp. 1606–1621.

Soldani, Siegfried et al. (2006). "Intermittent fault detection through message exchanges: a coherence based approach". In: *IFAC Proceedings Volumes* 39.13.

Steinwart, Ingo and Andreas Christmann (2008). *Support vector machines*. Springer Science & Business Media.

Sterten, Jo and Yurii Furtat (2017). "Regularized Methods of Noisy Signals Differentiation in Real Time". In.

Stevens, Trevor (2016). "GPlearn". In: *Github*. URL: (https://gplearn.readthedocs.io/en/stable/intro.html).

Subias, Audine and Louise Travé-Massuyes (2006). "Discriminating qualitative model generation from classified data". In: *20th International Workshop on Qualitative Reasoning (QR-06)*, pp. 129–136.

Svärd, Carl et al. (Dec. 2011). "A Data-Driven and Probabilistic Approach to Residual Evaluation for Fault Diagnosis". In: *Proceedings of the IEEE Conference on Decision and Control*, pp. 95–102. DOI: 10.1109/CDC.2011.6160714.

Tax, D.M.J. and R.P.W. Duin (2002). "Using two-class classifiers for multiclass classification". In: *2002 International Conference on Pattern Recognition*. Vol. 2, 124–127 vol.2.

Travé-Massuyès, L., T. Escobet, and X. Olive (2006). "Diagnosability Analysis Based on Component-Supported Analytical Redundancy Relations". In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 36.6, pp. 1146–1160. DOI: 10.1109/TSMCA.2006.878984.

Udrescu, Silviu-Marian and Max Tegmark (2020). "AI Feynman: A physics-inspired method for symbolic regression". In: *Science Advances* 6.16, eaay2631.

Uhlmann, Eckart et al. (2018). "Cluster identification of sensor data for predictive maintenance in a Selective Laser Melting machine tool". In: *Procedia manufacturing* 24, pp. 60–65.

Van Breugel, Floris, J. Nathan Kutz, and Bingni W. Brunton (2020). "Numerical Differentiation of Noisy Data: A Unifying Multi-Objective Optimization Framework". In: *IEEE Access* 8, pp. 196865–196877. DOI: 10.1109/ACCESS.2020.3034077.

Virgolin, Marco and Solon P. Pissis (2022). *Symbolic Regression is NP-hard*. arXiv: 2207.01018 [cs.NE].

Voydie, Dorian et al. (2023). "Machine Learning Based Fault Anticipation for 3D Printing". In: *22nd World Congress of the International Federation of Automatic Control (IFAC 2023)*.

Wang, Chengcheng et al. (2020). "Machine learning in additive manufacturing: State-of-the-art and perspectives". In: *Additive Manufacturing* 36, p. 101538.

Webb, Geoffrey I, Eamonn Keogh, and Risto Miikkulainen (2010). "Naive Bayes." In: *Encyclopedia of machine learning* 15.1, pp. 713–714.

Yang, Li and Abdallah Shami (2020). "On hyperparameter optimization of machine learning algorithms: Theory and practice". In: *Neurocomputing* 415, pp. 295–316.

Yang, Qingsong (2004). *Model-based and data driven fault diagnosis methods with applications to process monitoring*. Case Western Reserve University.

Yegnanarayana, Bayya (2009). *Artificial neural networks*. PHI Learning Pvt. Ltd.

Yen, Chih-Ta and Ping-Chi Chuang (2022). "Application of a neural network integrated with the internet of things sensing technology for 3D printer fault diagnosis". In: *Microsystem Technologies* 28.1, pp. 13–23.

Zerilli, John (2022). "Explaining machine learning decisions". In: *Philosophy of Science* 89.1, pp. 1–19.

Zhang, Shaohui et al. (2021). "Pre-classified reservoir computing for the fault diagnosis of 3D printers". In: *Mechanical Systems and Signal Processing* 146, p. 106961.

Zheng, Alice and Amanda Casari (2018). *Feature engineering for machine learning: principles and techniques for data scientists*. " O'Reilly Media, Inc."