



HAL
open science

Facilitating human inclusion in the data science process : from capturing business requirements to designing an operational machine learning workflow

Yassine El Amraoui

► To cite this version:

Yassine El Amraoui. Facilitating human inclusion in the data science process : from capturing business requirements to designing an operational machine learning workflow. Artificial Intelligence [cs.AI]. Université Côte d'Azur, 2024. English. NNT : 2024COAZ4017 . tel-04618657

HAL Id: tel-04618657

<https://theses.hal.science/tel-04618657>

Submitted on 20 Jun 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE DE DOCTORAT

Faciliter l'inclusion humaine dans le processus de science des données : de la capture des exigences métier à la conception d'un workflow d'apprentissage automatique opérationnel

Yassine EL AMRAOUI

Laboratoire d'Informatique, de Signaux et Systèmes de Sophia Antipolis (I3S)
UMR7271 UCA CNRS

Présentée en vue de l'obtention du grade de docteur en Informatique
d'Université Côte d'Azur

Dirigée par : Mireille BLAY-FORNARINO,
Professeur des Universités, Université Côte d'Azur

Co-dirigée par : Frédéric PRÉCIOSO, Professeur des Universités, Université Côte d'Azur

Soutenue le : 14/05/2024

Devant le jury, composé de :

Jean-Michel BRUEL, Professeur des Universités, Université de Toulouse

Anne ETIEN, Professeur des Universités, Université de Lille

Romain ROUVOY, Professeur des Universités, Université de Lille

Fabrice HUET, Professeur des Universités, Université Côte d'Azur

Julien MULLER, CTO, EZAKO

**FACILITER L'INCLUSION HUMAINE DANS LE PROCESSUS DE
SCIENCE DES DONNÉES : DE LA CAPTURE DES EXIGENCES MÉTIER
À LA CONCEPTION D'UN WORKFLOW D'APPRENTISSAGE
AUTOMATIQUE OPÉRATIONNEL**

*Facilitating Human Inclusion in the Data Science Process: From
capturing business requirements to designing an operational machine
learning workflow*

Yassine EL AMRAOUI



Jury :

Rapporteurs

Jean-Michel BRUEL, Professeur des Universités, Université de Toulouse
Anne ETIEN, Professeur des Universités, Université de Lille

Examineurs

Romain ROUYOY, Professeur des Universités, Université de Lille
Fabrice HUET, Professeur des Universités, Université Côte d'Azure

Directeur de thèse

Mireille BLAY-FORNARINO, Professeur des Universités, Université Côte d'Azure

Co-directeur de thèse

Frédéric PRÉCIOSO, Professeur des Universités, Université Côte d'Azure

Membres invités

Julien MULLER, CTO, EZAKO

Yassine EL AMRAOUI

Faciliter l'inclusion humaine dans le processus de science des données : de la capture des exigences métier à la conception d'un workflow d'apprentissage automatique opérationnel

last@ast@front
xiii+130 p.

Faciliter l'inclusion humaine dans le processus de science des données : de la capture des exigences métier à la conception d'un workflow d'apprentissage automatique opérationnel

Résumé

Le processus de création de workflows d'apprentissage automatique par les data scientists implique de comprendre les besoins métier, d'analyser les données et d'expérimenter pour trouver des solutions. Cependant, cette approche par essais et erreurs manque de structure pour prendre en compte toutes ces dimensions, et conduit souvent à des essais inadaptés et à des interprétations limitées des résultats par les praticiens. Pour remédier à ce problème dans le cadre de la conception de workflows destinés à la détection d'anomalies dans les séries temporelles, notre travail propose trois contributions principales :

- Prise en compte des données, des exigences métiers et des composants de la solution dans la conception des workflows de machine learning
- Personnalisation des workflows pour des solutions sur mesure en s'appuyant sur des descriptions partielles et modulaires du problème sous forme de configurations.
- Enrichissement de la connaissance portée par les lignes de produits logiciels par l'exploitation des produits. Ces contributions visent à rationaliser la création de workflows d'apprentissage automatique, en promouvant une approche plus structurée, le partage des connaissances et un meilleur alignement sur l'analyse du problème avant de rechercher des applications similaires.

Mots-clés : Ligne de produit logiciel, Science des données, Inclure l'humain dans la boucle

Facilitating Human Inclusion in the Data Science Process: From capturing business requirements to designing an operational machine learning workflow

Abstract

The process of creating machine learning workflows by data scientists involves understanding business needs, analyzing data and experimenting to find solutions. However, this trial-and-error approach lacks the structure to take all these dimensions into account, and often leads to inadequate testing and limited interpretations of results by practitioners. To address this problem in the context of designing workflows for anomaly detection in time series, our work proposes three main contributions:

- Consideration of data, business requirements and solution components in the design of machine learning workflows.
- Customization of workflows for tailor-made solutions, based on partial and modular descriptions of the problem in the form of configurations.
- Enriching the knowledge carried by software product lines through product exploitation.

These contributions aim to streamline the creation of machine learning workflows, promoting a more structured approach, knowledge sharing and better alignment with problem analysis before searching for similar applications.

Keywords: Software product line, Data science, Data science, Human in the loop

Acknowledgment

I would like to express my deepest gratitude to my esteemed mentors, Mireille Blay-Fornarino and Frederic Precioso, whose unwavering dedication, insightful guidance, and continuous support have been instrumental throughout my PhD journey. Their expertise, encouragement, and patience have not only shaped my research but have also inspired me to strive for excellence. I am truly fortunate to have had the opportunity to work under their supervision.

I extend my heartfelt appreciation to Anne Etien and Jean-Michel Bruel for their generosity in accepting the responsibility of examining my work and providing invaluable feedback as rapporteurs. Their constructive criticism and valuable insights have significantly enriched the quality of my research.

Special thanks are also due to Romain Rouvoy and Fabrice Huet for their willingness to lead my individual follow-up committee and serve as reviewers during my PhD defense. Their thorough evaluation and insightful comments have contributed immensely to the refinement of my work, and I am grateful for their invaluable contribution.

I am deeply indebted to Julien Muller for his unwavering support and encouragement at Ezako throughout the duration of my PhD. His mentorship and friendship have been invaluable, and I am grateful for the opportunity to have collaborated with him.

I extend my sincere gratitude to Philippe Collet for his collaboration and assistance in the development of the two publications associated with my research. His expertise and guidance have been invaluable assets to my academic endeavors.

I would like to express my appreciation to Anne Marie Dery-Pinna for her steadfast support and encouragement throughout my PhD journey. Her mentorship and belief in my abilities have been a source of motivation during challenging times.

I am also grateful to Thomas Jalabert, Jonathan Behagel, Edoardo Savini, and Cyrus Boumedine for their camaraderie and support at Ezako. Their friendship and encouragement have made my PhD experience more enriching and enjoyable.

Additionally, I extend my thanks to Nicolas Lacroix and Yann Brault for their dedication and contribution to the sub-parts of my research during their internships. Their hard work and commitment have significantly contributed to the advancement of my research objectives.

Finally, I would like to express my deepest gratitude to my family, especially my parents and my brother, for their unwavering love, encouragement, and support throughout my academic journey. Their sacrifices and belief in my abilities have been a constant source of strength and motivation. I am profoundly grateful for their unconditional support and guidance, which have been instrumental in shaping my academic and personal growth.

Table of contents

1	Introduction	1
1.1	Ezako	1
1.2	General context	1
1.3	Diving into Data science ecosystem	2
1.4	Example: Anomaly detection on time series of bearing motor currents in a washing machine	3
1.4.1	First iteration: Initial enquiry	3
1.4.2	Second iteration: Refining data understanding	4
1.4.3	Third iteration: Refining understanding of business requirements	5
1.4.4	Synthesis	5
1.5	Manuscript overview	6
	Vocabulary	7

State of the art

2	Sharing data science experience	11
2.1	Characterizing an ML problem	13
2.2	Data scientists practices	13
2.3	Characterizing Diversity in Machine Learning Solutions	14
2.4	Domain Experts are inherent drivers of data science projects	14
2.5	Requirements in ML	15
2.6	Methodological approaches to machine learning	16
2.6.1	CRISP-DM	16
2.6.2	AutoML	17
2.6.3	Synthesis	19
3	Sharing data science solutions	21
3.1	Sharing machine learning implementations	23
3.1.1	R	23
3.1.2	Weka (Waikato Environment for Knowledge Analysis)	23
3.1.3	Knime (Konstanz Information Miner)	28
3.2	Sharing data science experiments	30
3.2.1	Reuse of notebooks by clone-and-ow	30
3.2.2	OpenML	31
3.2.3	KNIME Hub	32
3.3	Synthesis	33

4	Software product lines	35
4.1	Introduction	37
4.2	Feature models	37
4.3	Lopez-Herrejon and Batory standard example	38
4.4	SPL's for composing scientific workflows	39
4.5	Feature models edition	40
4.6	Evolution-aware evaluation	41
4.7	Synthesis	44
Contributions		
5	Capturing Knowledge from trial and error	47
5.1	Introduction	49
5.2	Experiments design	52
5.2.1	Scenarios description	52
5.2.2	Summary table	54
5.2.3	Results description	56
5.2.4	Discussion	58
5.2.5	Explicated knowledge	59
5.3	Conclusion	60
6	Contextualizing ML variability modeling	61
6.1	Introduction	63
6.2	Approach overview	64
6.2.1	Contextualized Solutions within the Overall Solution Space	64
6.2.2	Process overvie	66
6.2.3	Tooling overvie	67
6.3	Contextualized ML Solution Variability Modeling	68
6.3.1	Multi-Domain Variability Modeling: Supporting Contextualized Solution Space Reduction	68
6.3.2	Modeling Applications and Code Interactions for Efficient Solution Retrieval	70
6.3.3	Formal definition of the interaction	71
6.4	Applications	72
6.4.1	Scenario 1: Retrieve and clone notebook	72
6.4.2	Scenario 2: Generate a new notebook	73
6.4.3	Industrial case stud	73
6.5	Conclusion	78
7	Accommodating the incremental acquisition of knowledg	79
7.1	Introduction	81
7.2	Identifying evolution smells to uncover implicit knowledge	82
7.2.1	Formalization	83
7.2.2	Pattern detections	85
7.2.3	Limitations and Future Directions	85
7.3	Knowledge assessment metrics	86

7.3.1	Domain-Level Feature Model Metrics	86
7.3.2	Evaluating Feature Model through the Lens of Produced Products	86
7.3.3	Limitations and Future Directions	88
7.4	Evolving Configurations to Align with Feature Model Evolution	88
7.4.1	Application-preserving refactoring against practice evolutio	88
7.4.2	Enhancing Knowledge Through the Integration of New Application	89
7.4.3	Limits and Future Directions	89
7.5	Applications	90
7.5.1	First three steps of the SPL construction process	90
7.5.2	Knowledge extraction driven by SPL assessment	91
7.5.3	Knowledge extraction driven by SPL evolution assessment	92
7.5.4	Exploiting the metrics	93
7.6	Conclusion	94
Conclusion & perspectives		
8	Conclusion and Perspectives	97
8.1	Conclusion	97
8.2	Perspectives	98
8.2.1	FATE	98
8.2.2	Using LLMs to conduct the domain expert interview	101
	References	107
	List of figures	119
	List of definitions	121
	List of examples	123

Appendices

CHAPTER 1

Introduction

chap_intro

1.1 Ezako

Ezako stands as a specialized company dedicated to crafting machine learning (ML) solutions tailored for detecting anomalies in time series data, addressing the unique needs of its clients. Within an ML project, the role of a data scientist is pivotal, involving tasks such as understanding data intricacies, grasping business requirements deeply, and formulating solutions aligned with chosen Key Performance Indicators (KPIs). This necessitates collaborative engagements with domain experts, project managers, as well as fellow data scientists and ML engineers.

Each ML project at Ezako follows a structured approach with multiple iterations, starting each iteration by conducting an interview with the domain expert. This interview aims to glean insights into specific business requirements, serving to validate the modeling phase and identify undesirable behaviors in the solution. Currently, it falls upon the data scientist to actively seek actionable information from the domain expert, conduct comprehensive data analysis, and make informed decisions on the modeling approach for every new project.

Consequently, the choice of modeling techniques and engineering decisions becomes highly reliant on the individual addressing the problem, introducing potential variations and increasing diversity among generated solutions. Additionally, analyzing business requirements poses a non-trivial challenge, demanding the mapping of business needs from diverse domains of expertise to actionable ML requirements. This complexity adds to the variability, making each new problem often necessitate starting from scratch. This underscores the substantial effort and adaptability required to address these challenges effectively.

1.2 General context

My research is conducted within the CIFRE framework, entailing a collaborative endeavor involving myself as the Ph.D. candidate, a research laboratory, and a corporate partner. The laboratory provides expertise in software engineering and artificial intelligence (AI), while the company grants access to real projects, offering insights into the workings of a machine learning initiative. My role in this collaboration was bridging the gap between the company's requirements and the expertise required for implementation.

After an extensive period devoted to addressing the challenge of guiding data scientists in designing machine learning workflows, with a specific constraint to deviate from automated machine learning (autoML) I came up with the following questions:

1. How do data scientists proceed to reduce the solution space?
2. How do data scientists share machine learning case studies?

3. How to share acquired knowledge about the current state of machine learning practices?

Since the company's focus is on time series anomaly detection, this work focus on application in that context. We identify the following functional requirements.

R1- Identifying similarities between partially described problems. Data scientists naturally initiate their exploration by seeking resemblances and distinctions among past problems. However, this project is laden with intricacies. The intrinsic nature of source data frequently imparts a complex characterization, particularly in the context of precisely delineating anomalies within temporal data sequences. This intricacy extends to data providers, whose efforts to pinpoint anomalies can also be challenging. Efficient processing of partially characterized source data plays a key role. This encompasses the formulation of algorithms with the capability to navigate the diverse inherent in datasets. A pertinent instance arises when the precise archetype of anomaly remains elusive (such as isolated instances, pervasive anomalies, or distinct patterns (Chandola, Banerjee, & Kumar, 2009)).

R2-Consolidating knowledge according to the evolution of practices. When we consider new solutions, like different ways of using machine learning, we need to understand the specific problems they are meant to solve. It is not just about picking new algorithms; we also have to think about things like how to measure success and what the business needs. For example, finding unusual patterns in data collected from IoT systems affects how we apprehend the whole machine learning process.

To build a strong understanding, it is important to compare different ways of solving problems. By "ways of solving problems," we mean not only the solutions and how well they work, but also the actual problems themselves, including the data and what the business wants to achieve. What is particularly interesting is finding similar problems that end up with different solutions. This can help us discover new insight we should consider, or realize that some old ways of thinking may not be relevant anymore. For instance, if two different sets of tools work well on the same data, we want to figure out why, maybe it is because of some specific things the data hold or the business requires.

1.3 Diving into Data science ecosystem

The data scientist ecosystem operates as a collaborative network, comprising domain experts, data scientists, and ML engineers collaborating to address intricate challenges within the domain of data science and machine learning.

Central to this ecosystem are domain experts, individuals with knowledge in specific fields like finance, healthcare, or marketing and so on. These experts offer crucial insights into the intricacies of their domains, providing contextual understanding and domain-specific requirements crucial for effective ML solutions.

In this collaborative setting, data scientists play a pivotal role, tasked with understanding data intricacies, extracting meaningful patterns, and formulating models aligning with identified business requirements. Leveraging their statistical and analytical proficiencies, data scientists bridge the gap between domain knowledge and technical implementation, ensuring accuracy and relevance in the developed solutions.

Collaboration extends to ML engineers, possessing specialized knowledge in implementing ML models and systems. ML engineers work closely with data scientists to translate algorithms

1.4 – Example: Anomaly detection on time series of bearing motor currents in a washing machine

into functional, scalable solutions, contributing expertise in software development, system architecture, and infrastructure optimization for efficient project deployment.

This ecosystem thrives on frequent interactions and knowledge exchange among stakeholders. Domain experts provide critical feedback during project iterations, ensuring ML solutions meet distinct business requirements. Data scientists collaborate, sharing insights, best practices, and innovative approaches, fostering a culture of continuous learning. ML engineers contribute technical expertise, supporting the effective implementation and operationalization of data scientists' models.

The data scientist ecosystem heavily relies on day-to-day interactions and collaborative exchanges among domain experts, data scientists, and ML engineers as the primary mechanism for harnessing collective knowledge and skills. These informal interactions signify valuable information sharing, including reusable solutions from past applications see fig. 1.1.

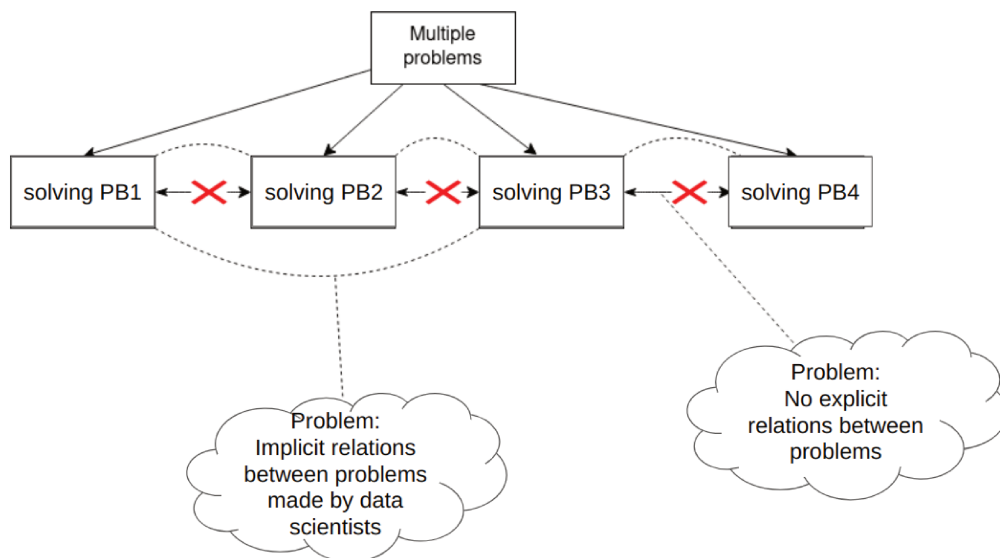


Figure 1.1: The relations between multiple ML problems

fig:fig_multipb

1.4 Example: Anomaly detection on time series of bearing motor currents in a washing machine

1.4.1 First iteration: Initial enquiry

1. The expert defines the final objective: detection of anomalies in the data.
2. The data is unlabeled.
3. The data scientist chooses to use an unsupervised One-class SVM.
4. Result: Failure to locate anomalies of interest to the expert.

The data scientists presented the spikes as anomalies in this iteration see fig. 1.2. The expert rejected them as they were acquisition errors. So the data scientist asked the expert to label what he considers anomalies for the second iteration.

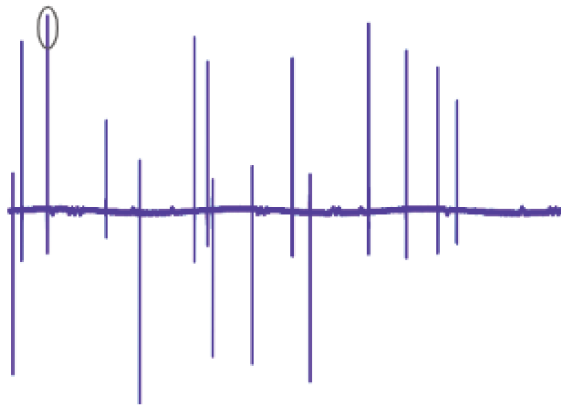


Figure 1.2: Acquisition errors

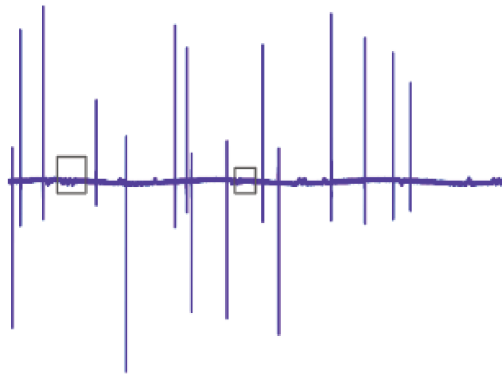
`fig:spikes`

Figure 1.3: Expert annotations

`annotations`

1.4.2 Second iteration: Refining data understanding

1. The expert labels part of the data see fig. 1.3
2. The data scientist chooses a pool of algorithms to compare based on a supervised evaluation metric
3. The data scientist chooses a ResNet based on its performance
4. Result: the ResNet localized the anomalies the expert wants
5. The model is deployed in production
6. But in production the ResNet does not localize the anomalies

After the ResNet model performed acceptably on the test set it was deployed in production, but in production the expert discovered that the model didn't localize anomalies. He contacted the data scientist and presented the issue and this time was able to share the data in production with the data scientist.

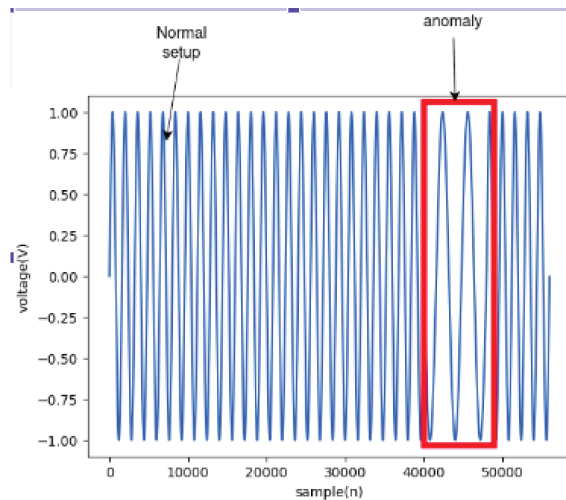


Figure 1.4: Drifts in the data

`fig:drifts`

1.4.3 Third iteration: Refining understanding of business requirements

1. Upon examining the data, the data scientist understood that the magnitude of the frequencies distortions in the production time series were different from the test. There is a drift (a change in the anomalies properties over time) in the data see fig. 1.4.
2. The data scientist should have asked whether current frequencies could drift in production.
3. The data scientist used a semi supervised CNN AutoEncoder
4. Result: the anomalies were localized in production

After confirming with the expert the existence of drifts in the anomalies properties, the data scientist made a connection with a past use case of anomaly detection in sounds and reused the same workflow which was based on a CNN AutoEncoder.

1.4.4 Synthesis

In this example, the data scientist needed to characterize the problem both in terms of data and solution through interaction with the domain expert. Once characterized, he established a workflow based on his preferences during the first and second iterations. In the third iteration, he connected it to a previous problem and reused its solution. The questions that arise from this procedure are:

1. What characterizes an ML problem?
2. What characterizes an ML solution?
3. How does the domain expert influence the process of designing an ML workflow?
4. Does the data scientist follow a specific methodology in the analysis of the problem and the design and evaluation of the solution?
5. How do data scientists share solutions?

On a side note, this process was driven by a specific data scientist, meaning that if another data scientist were in charge, the number of iterations could vary. More iterations might be needed if the scientist had to look up the problem in the literature, or fewer iterations could occur if the drift nature of the anomalies was clarified from the beginning. The process is quite random and depends on individual data scientists. Therefore, additional questions arise:

1. How can we standardize the process of analyzing ML problems?
2. How can we standardize ML solutions?

Even if we manage to standardize problem analysis and solution design, we cannot hope to capture all existing knowledge about time series anomaly detection, let alone generalize it to all machine learning. Therefore, this shared knowledge that drives problem analysis and solution design will need to evolve. The questions that arise are:

1. How can we ensure that knowledge about the composition of a scientific workflow evolves?
2. How can we evaluate this evolution?

We will address all these questions in the state-of-the-art section of this manuscript.

1.5 Manuscript overview

This manuscript is divided into two main parts: the state of the art and the contributions with their validation. The state of the art consists of three chapters, covering topics such as the sharing of data science experience, data solution sharing, and modeling scientific workflows using software product lines. The contributions part also comprises three chapters: the first addresses the effective capture of knowledge from the trial and error process undergone by data scientists, the second demonstrates how to model the variability of the data science domain according to its context, and the third showcases methods for accommodating knowledge evolution in the model, built in the previous chapter and extracting new knowledge from these evolutions.

Vocabulary

Définition 1.5.1 (Software product lines). "Software product-lines are a paradigm for managing the complexity of tracking and creating reusable software artifacts, as well as describing their points of variability, and ensuring they are reused appropriately. A key part of an SPL is scope, commonality, and variability (SCV) analysis. The scope defines the collection of software artifacts that constitute the SPL." (Rosenmüller & Siegmund, 2010)

Définition 1.5.2 (Commonality). "The commonality defines the attributes that are common across different sets of artifacts." (Rosenmüller & Siegmund, 2010)

Définition 1.5.3 (Variability). "The variability describes the differences that exist across the artifacts, such as various implementations and algorithms for different environments and/or requirements" (Rosenmüller & Siegmund, 2010)

`def_configSPL1` **Définition 1.5.4** (Configuration). "A configuration of a product line is a set of selected features." (Nieke et al., 2022)

Définition 1.5.5 (SPL and Feature Model). A software product line $SP Li$ is a set of products (e.g., software services) described by a feature model $F Mi$. Each product of $SP Li$ is a combination of features and corresponds to a valid configuration of $F Mi$. A configuration c of $F Mi$ is defined as a set of features selected, i.e., $c = f_1, f_2, \dots, f_m$ with f_1, f_2, \dots, f_m features of $F Mi$. $JF MiK$ denotes the set of valid configurations of the feature model $F Mi$ (Acher et al., 2012)

`def_configSPL` **Définition 1.5.6** (configuration). In this chapter when we refer to configurations it is a software product line configuration as mentioned in definition 1.5.4. However those configurations contain a description of the ML problem and data as long as the solution or workflow used to solve the machine learning application or use case

`def_subspace` **Définition 1.5.7** (sub-space). The term "sub-space" refers to a restricted or specific subset of problems, solutions, criteria, or artifacts within a broader domain, particularly in the realm of machine learning (ML) experiments and applications .

`def_partialConf` **Définition 1.5.8** (partial configuration). Partial configuration refers to a specific configuration of a software product that includes only a subset of the available features and options defined by the product line.

`def_domainSPL` **Définition 1.5.9** (a domain). The domain in feature models defines the specific field of application or context for which the feature model is being developed, helping to organize and structure the configurable elements of the software system within that context.

`def_solSPACE` **Définition 1.5.10** (solution space). The solution space in the context of data science refers to the entire range of possible solutions that a data scientist explores and evaluates when working on a problem. It encompasses all the potential approaches, methods, algorithms, parameters, and techniques that could be employed to address a specific challenge or task.

partialConf

Remark 1.5.1 – Our grasp of knowledge remains constrained within a certain temporal frame. This inherent limitation restrains our ability to encompass all potential aspects essential for problem characterization or to account for the entirety of existing machine learning solutions. Consequently, our configuration depiction resembles a canvas of restricted dimensions, open to expansion upon the unearthing of novel criteria or integration of additional machine learning approaches. Thus, any configuration, while seemingly comprehensive within the present scope of feature attributes, persists as an incomplete representation, failing to encompass the entirety of potential criteria.

i.e., At any given time, our understanding is limited. This means we can't describe a problem using all the possible details or consider every existing machine learning solution. So, when we represent a setup or arrangement, it's like drawing a picture on a small piece of paper. We can add more to it later when we learn new things or find new solutions. So, no matter how complete a setup seems based on what we know now, it's still not complete because we haven't thought about all the possible details yet.

application

Remark 1.5.2 – we use the term "application" as an additional designation for "ML use case."

productSPL

Définition 1.5.11 (product). "product" refers to a specific instance or variant of a software system that is derived from the common assets, features, and configurations defined within the product line. These common assets serve as a foundation, and various products can be generated by selecting specific features and configurations from the available options.

icationprod

Définition 1.5.12 (Application products). An Application product is a combination of features, business requirements, data, and machine learning elements that define a successfully addressed machine learning scenario. In simpler terms, it comprises the characteristics outlining both the ML problem and its solution.

MLWFproduct

Définition 1.5.13 (*ML workflow product*). When we mention an ML workflow product, we're talking about a solution to a specific problem. It is the dedicated process that uses machine learning techniques to solve the problem. This process includes not only the machine learning algorithms but also the steps to evaluate, monitor and maintain the solution's effectiveness in real-world use over time.

State of the art

CHAPTER 2

Sharing data science experience

2.1	Characterizing an ML problem	13
2.2	Data scientists practices	13
2.3	Characterizing Diversity in Machine Learning Solutions	14
2.4	Domain Experts are inherent drivers of data science projects	14
2.5	Requirements in ML	15
2.6	Methodological approaches to machine learning	16
2.6.1	CRISP-DM	16
2.6.2	AutoML	17
2.6.3	Synthesis	19

2.1 Characterizing an ML problem

The perspectives presented in (Horkoff, 2019; Vogelsang & Borg, 2019) collectively emphasize the multifaceted criteria for characterizing Machine Learning (ML) problems. Functionally, these problems necessitate precise task delineation, spanning classification, regression, clustering, or recommendation in conjunction with precise specifications of input data sources and expected outcomes. Both texts stress the critical need for a nuanced understanding of performance metrics, tailoring them to the problem domain to ensure the solution considers business requirement functional and non functional.

In harmony with functional criteria, Horkoff ^{horkoff2019non}underscores the significance of non-functional considerations, extending beyond data-centric elements(Horkoff, 2019) to satisfy stakeholder-specific requirements (*i.e.*, business requirements or non-functional requirements). These encompass dimensions such as fairness, transparency, security, privacy, and testability. Fairness is elucidated as ensuring equitable treatment by ML models, transparency as facilitating interpretable decision-making processes, security as fortifying systems against unauthorized access, privacy as safeguarding sensitive data, and testability as ensuring robust evaluation methodologies.

TAW2dimensions

Takeaway 1. *An ML problem is characterized by two dimensions, the data-centric dimension and the stakeholder’s business requirements; each dimension can be decomposed into many criteria.*

2.2 Data scientists practices

To enhance the robustness and performance of their machine learning solutions, data scientists employ iterative testing and experimentation strategies (Kim, Zimmermann, DeLine, & Begel, 2017). These strategies involve a cyclical process of hypothesis formulation, testing, analysis of results, and subsequent refinement.

In addition to this iterative process, data scientists continuously adapt and refine their solution development practices based on feedback from previous projects (Kross & Guo, 2019). Each project is a learning opportunity, enabling data scientists to fine-tune their methodologies and strategies. This adaptive nature ensures that data science remains responsive to evolving challenges and changing data landscapes.

Within the field of data science, solutions-sharing practices play a crucial role. Data scientists often collaborate and share their solutions within their teams and across the organization (Epperson, Wang, DeLine, & Drucker, 2022). Sharing encompasses both code and insights gained from previous analyses. The practice of sharing code ranges from personal analysis reuse to the development of team-wide shared repositories.

Moreover, data scientists share template notebooks and shared libraries to facilitate the sharing of their machine learning models and data processing pipelines (Kim et al., 2017). Template notebooks are pre-configured and documented analysis environments that enable data scientists to create reusable, customizable solutions. Shared libraries contain commonly used functions and methods that enhance the efficiency and consistency of data science projects.

TAW-DSPRACTICES

Takeaway 2. *Data scientists use iterative testing and experimentation strategies to continually adapt their practices based on feedback; they resort to sharing solutions by employing tools such as template notebooks and shared libraries as facilitators for data science practice sharing.*

2.3 Characterizing Diversity in Machine Learning Solutions

Diversity plays a pivotal role in enhancing the performance and reliability of machine learning (ML) solutions. Understanding how to characterize diversity within ML solutions is a critical aspect of improving model generalization and robustness. This diversity is decomposed into various dimensions, including the diversity originating from different models, preprocessing techniques, and workflows (Gong, Zhong, & Hu, 2019)

One fundamental source of diversity in ML solutions arises from employing different models (Fernández-Delgado, Cernadas, Barro, & Amorim, 2014). Researchers and practitioners often experiment with various algorithms, architectures, and hyperparameters to tackle specific tasks for instance the scikit learn library presents the implementations of 61 supervised classifier, 9 clustering models.

Beyond model diversity, preprocessing techniques are another key contributor to diversity within ML solutions. These techniques encompass data transformations, feature engineering, and data augmentation for instance scikit-learn provides 16 preprocessing implementations. By applying diverse preprocessing methods, ML practitioners can extract varied representations from the same data, enabling models to capture different facets of the information.

Diversity in ML solutions is not limited to models and preprocessing; it also extends to the workflows themselves. ML workflows can also differ significantly in terms of data collection, labeling, and model deployment strategies. This diversity in workflows is essential, as it ensures that ML solutions are adaptable to different application domains and constraints (Xin, Ma, Song, & Parameswaran, 2018).

LargeDomain

Takeaway 3. *The diversity in machine learning solutions arises due to inherent variability across the entire process, encompassing algorithm selection, data preparation, and workflow design. This diversity is amplified by the individual modeling decisions of data scientists, leading to distinct outcomes.*

mainExperts

2.4 Domain Experts are inherent drivers of data science projects

Data science projects involve collaboration both within the data science team and with domain experts (Piorkowski et al., 2021; A. X. Zhang, Muller, & Wang, 2020). Because even though data does not lie, it does not tell the whole truth.

Throughout various stages of the modeling process, including data access, feature extraction, evaluation, and result communication, domain experts play a pivotal role (Piorkowski et al., 2021). However, these collaborative activities currently face challenges, as there is insufficient support, resulting in data scientists having to rely predominantly on their intuitive understanding of their data (Muller et al., 2019; Passi & Jackson, 2018; Mao et al., 2019).

While emerging tools aim to address technology gaps in supporting collaborative data science practices, they often prioritize data science teams, potentially leaving domain experts with limited involvement (A. Y. Wang, Mittal, Brooks, & Oney, 2019).

expertNeeded

Takeaway 4. *In data science projects, collaboration between data scientists and domain experts is standard but often time constrained, limiting domain experts' participation. To maximize experts' contribution, improved methodological support and tools are needed at each project stage.*

2.5 Requirements in ML

In the realm of Software Engineering (SE) and precisely systems and requirements engineering, a non-functional requirement (NFR) refers to a stipulation that outlines criteria for evaluating the performance of a system, rather than dictating precise behaviors. These differ from functional requirements (FRs), which define specific functions or actions. While the blueprint for realizing functional requirements is articulated in the system design, the blueprint for fulfilling non-functional requirements is typically elaborated upon in the system architecture, as they tend to have a significant impact on the system's architectural aspects (Wada, Suzuki, & Oba, 2006).

In SE, Requirements (NFRs+FRs) are essential for providing a strong foundation for quality assessment (Chung, Nixon, Yu, & Mylopoulos, 2012). Attributes like speed, usability, and security are systematically documented, offering a roadmap for developers to follow (Glinz, 2007; Chung & do Prado Leite, 2009). Similarly, collecting and documenting of requirements within Machine Learning (ML) is essential but introduces a contrasting scenario characterized by empirical gathering of context-dependent requirements (Smola, 2008; Kamishima, Akaho, & Sakuma, 2011).

For instance, consider the functional requirement of fairness. In SE, fairness can be methodically addressed based on established guidelines, resulting in a relatively clear understanding. In contrast, within the ML domain, fairness takes on various interpretations depending on the application context. Defining fairness for healthcare differs significantly from fairness considerations in autonomous driving. This contextual adaptability is a central contrast between SE and ML, where the fluidity of requirements definitions becomes apparent (Kamishima et al., 2011; Ntoutsis et al., 2020).

In SE, requirements are subject to a rigorous understanding of how various components, including algorithms and optimizations, affect relevant qualities (Chung et al., 2012; Pereira et al., 2021). This understanding is obtained through accurately framing the requirements boundaries to allow for its systematic integration into the development process. However, the landscape shifts when we delve into the ML domain. Here, incomplete knowledge regarding the interactions between ML algorithms, optimizations, and assumptions introduces ambiguities regarding the translation of business requirements into actionable requirements ml wise (Mohassel & Zhang, 2017).

Unforeseen conflicts between fundamental attributes in machine learning systems, such as the trade-offs between accuracy and processing speed, add further complexity to the situation. These unexplored areas underscore the distinction between software engineering (SE) and machine learning (ML) when it comes to defining requirements, as highlighted by (Mohassel & Zhang, 2017). Software engineers generally have the advantage of being thought that these requirements must be formalized and documented for mutual exploitation. In contrast, ML practitioners often grapple with nuanced uncertainties on an individual basis, stemming from the many different existing intricacies of ml workflows and real-world applications (Kamishima et al., 2011; Ntoutsis et al., 2020).

SE primarily involves selecting requirements for implementation, resulting in a well-established process (Chung et al., 2012). Similarly, ML introduces its specific challenges: it necessitates decisions concerning algorithm types, characteristics, assumptions, training data, and optimizations, based on ML technical understanding but also the understanding of the domain of application making requirements specification complex (Horkoff, 2019). Unlike SE, where the primary question is "which requirements do I implement?" ML prompts a different query:

"How do business requirements constrict the algorithm type, the assumptions on data distribution, training data, optimizations and so on?" (Horkoff, 2019).

This shift in focus highlights the contrast between SE and ML in requirements specification. SE thrives on structured requirements, whereas ML should require a more adaptive and context-aware approach hardly generalized, reflecting the diversity and dynamism of machine learning applications (Kamishima et al., 2011; Ntoutsis et al., 2020). Moreover, ML systems operate in a constant state of evolution, adapting to new data and changing quality requirements (Chung et al., 2012). Therefore, requirement engineering in ML necessitates documenting and adapting requirements as systems evolve (Kamishima et al., 2011; Ntoutsis et al., 2020).

alKnowledge

Takeaway 5. *Managing business requirements in machine learning (ML) involves addressing the challenges of incomplete knowledge regarding how ML workflow selection is affected by business requirements, especially when working with fragmented and context-dependent definitions of these requirements, and accommodating the continuous evolution of ML systems driven by changing data and business requirements themselves. These complexities underline the need for a nuanced and adaptive approach to capturing business requirements in the dynamic field of machine learning.*

2.6 Methodological approaches to machine learning

ota_crispDM

2.6.1 CRISP-DM

CRISP-DM (Cross Industry Standard Process for Data Mining) helps drive the data science process, beyond just using algorithms (Wirth & Hipp, 2000). It identifies and organizes essential aspects that factor into the success of a data science project, from analyzing data, gathering the functional and non functional business requirements, and uncovering new knowledge (Brachman, 1996) in the case of classical data mining projects, or develop a machine learning workflow that satisfies the stakeholders business requirements given the exploited data.

CRISP-DM is widely used in many industries, like manufacturing, translating complex problems into doable data tasks (Schröder, Kruse, & Gómez, 2021). It adapts to different fields, connecting with the broader Knowledge Discovery process. CRISP-DM has six phases that work together to guide data mining, starting from understanding the problem throughout the process *i.e.*, Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation, Deployment (Wirth & Hipp, 2000).

In practice, data scientists have to carry on additional phases that are not formalized in the CRISP-DM methodology nor any KDD popular variants such as SEMMA. For instance, Huber, Wiemer, Schneider, and Ihlenfeldt proposed formalizing the phase of technical understanding for instance. Technical understanding as presented in (Huber et al., 2019) is the conversion of the business goals into actionable technical goals as well as gathering domain expertise related to developing an effective strategy for conducting experiments. However technical understanding on a methodological level can be assimilated to a sub-phase of business understanding, therefore the methodology does not lack the technical understanding phase but can not support the data scientists in implementing some of its phases such as business understanding.

huber2019dmme

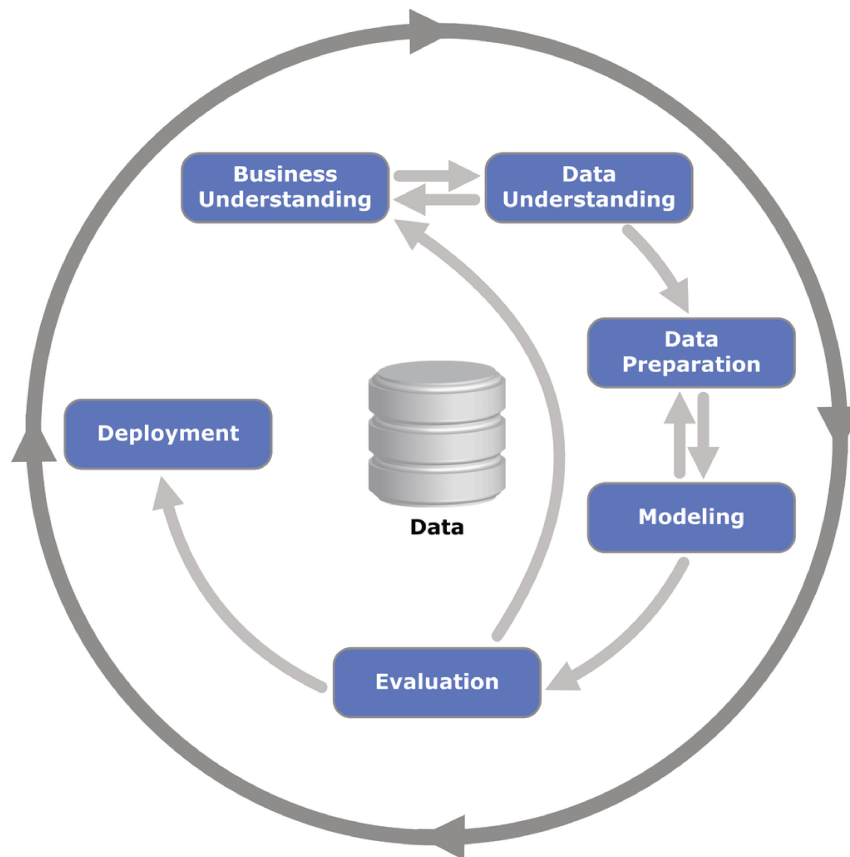


Figure 2.1: Process diagram showing the relationship between the different phases of CRISP-DM

TAW-Methodo

Takeaway 6. *In addition to the support at a methodological level, data scientists need additional support at the implementation level of certain phases and their sub-phases, such as business understanding and technical understanding.*

ss_AMLSOTA

2.6.2 AutoML

Automated machine learning (AutoML) automatically resolves machine learning tasks by taking the data scientist out of the loop. “Automated machine learning (AutoML) aims to reduce the demand for data scientists by enabling domain experts to build machine learning applications automatically without extensive knowledge of statistics and machine learning.” (Zöller & Huber, 2021). By feeding the data into an AutoML tool that solves the task they want, a domain expert can obtain a machine learning workflow optimized according to ML metrics such as accuracy, precision, f measure, ROC (Powers, 2020), or Matthew Correlation Coefficient (Yao et al., 2018). However, deciding which metric to use and which trade-offs to make (for instance, prioritizing detecting the true positives or minimizing the false negatives) requires understanding the domain problem, the data, and machine learning methods. For instance, using accuracy as an evaluation metric to solve a heavily unbalanced classification problem results in invalid biased estimators. Moreover, deciding between ROC, f-measure, and MCC for an unbalanced classification “depends on the situation” and has to be configured accordingly to the use-case. Such a configuration step

requires the data scientist both to understand the problem and the data, and to be included in the process alongside AutoML.

Furthermore, an AutoML tool resolves a subset of machine-learning tasks, "*The fully-automated setting that current AutoML systems operate on, may not be a one-size-fits-all solution for many users and problem domains*" (D. J.-L. Lee & Macke, 2020). Choosing a tool compatible with the use-case requires the user to understand the ML application this tool automates, *i.e.*, the data, the domain expert's requirements, and the appropriate machine-learning techniques to tackle it. We recognize two types of incompatibilities. The first is when available tools handle the ML problem part but not the data, while the second is when available tools do not handle the use-case (both problem and data).

Because AutoML can benefit from human intervention, recent AutoML human-guided approaches aim at capitalizing on human knowledge. They use domain expert knowledge to steer the data engineering part (Brownlee, 2020), data scientists' insight to restrict the workflow selection (Xin, Wu, Lee, Salehi, & Parameswaran, 2021) and the models' hyperparameter optimization processes. However, once the data scientists have understood the ML problems' use-case well enough to guide the workflow selection, they do not need full automation of the process. They need a suggestion system in order to support them during the model selection stage while maintaining control over the reason each ML component is used (D. Wang et al., 2021), and possibly automation of the hyperparameter optimization (HPO) phase.

Furthermore, even though we observe a growing trend in automating the ML workflow selection for some ML tasks, AutoML will not replace the data scientists in solving use-cases that deviate from the well-trod paths since they adapt and develop their own machine learning practices¹ or implicit knowledge to accommodate the specific requirements and constraints of these use-cases when AutoML can not.

As part of the human-centered AI research efforts, initial efforts have been made to formalize, through explicit modeling, the data scientists' implicit knowledge or practices, data scientists heavily rely on; similarly to software engineers, "*software development relies heavily on implicit knowledge*" (LaToza, Venolia, & DeLine, 2006). Through software product line modeling (Clements & Northrop, 2002), we believe that encouraging tools (Camillieri et al., 2016) and Frameworks (Amraoui, Blay-Fornarino, Collet, Precioso, & Muller, 2022) have emerged. For a given machine learning task, they aim to continuously support the data scientist in reducing the number of iterations necessary to solve a machine learning use-case accordingly to their prior knowledge. Furthermore, it aims at alleviating the workflow selection effort by systematically suggesting reductions of the search space on demand. Moreover, they attempt to explicit the criteria allowing to retrieve domain experts' definitions of their use-cases.

TAW-AutoML

Takeaway 7. *While AutoML is proficient in model selection, human intervention remains crucial, particularly in domains with specialized knowledge, complex problem-solving, and informed decision-making needs. Therefore, AutoML's success hinges on cooperation with human experts for effective machine-learning results. Similarly, as in traditional machine learning workflows, data scientists may need guidance in tool selection based on functional and non-functional requirements.*

¹by ML practices we mean untold rules and data analysis the data scientist relies on to design a machine learning workflow for a given application

2.6.3 Synthesis

An ML problem is characterized by two dimensions: the data-centric dimension and the business requirements-centric dimension. Therefore, co-constructing a solution is challenging due to the different vocabularies used by the expert and the data scientist, as well as the data scientist's limited understanding of the business problem. Additionally, iterative testing and experimentation strategies rely on expert feedback, while access to the expert is limited in both time and availability. *Therefore there is a need to capture the context of the problem in terms of business requirements and data properties as early as possible.*

CHAPTER 3

Sharing data science solutions

3.1	Sharing machine learning implementations	23
3.1.1	R	23
3.1.2	Weka (Waikato Environment for Knowledge Analysis)	23
3.1.3	Knime (Konstanz Information Miner)	28
3.2	Sharing data science experiments	30
3.2.1	Reuse of notebooks by clone-and-ow	30
3.2.2	OpenML	31
3.2.3	KNIME Hub	32
3.3	Synthesis	33

3.1 Sharing machine learning implementations

3.1.1 R

R (Team, 2000) is both a programming language and an environment that is widely used for statistical computing and data analysis. It is known for its flexibility, extensibility, and rich ecosystem of packages, making it a popular choice among data analysts, statisticians, and data scientists.

R offers efficient implementations of numerous machine learning algorithms, making it a formidable choice for data mining tasks (Ihaka & Gentleman, 1996). Notably, it includes popular algorithms such as decision trees, k-means clustering, and support vector machines, enhancing its versatility. It also includes visualization packages such as ggplot2 package that facilitates intricate and customized data visualizations. It introduces specific data types like data frames for handling large datasets and supports parallelization, enabling efficient data processing.

Despite its capabilities, R poses challenges. Its syntax and functional approach can be daunting for citizen data scientists (Merkelbach, Von Enzberg, Kühn, & Dumitrescu, 2022; Boehmke & Greenwell, 2019), with a steep learning curve. While basic tasks like creating plots and conducting descriptive statistics are manageable, harnessing R's full potential demands extensive practice even for confirmed developers (Shrestha, Botta, Barik, & Parnin, 2020).

Addressing the challenge of R's steep learning curve, the Rattle package emerges as a valuable solution. Developed since 2006, it offers a more user-friendly graphical user interface (GUI) (Williams et al., 2009) (see fig. 3.1). Rattle simplifies the workflow by selectively loading packages from R as needed for specific analyses. This approach streamlines tasks, such as building predictive models, enabling data scientists to focus on analysis rather than struggling with R's intricacies. Rattle see fig. 3.1 leverages some of R's standard data mining algorithms. In the realm of data transformations it provides rescalings, imputing (which is filling in the gaps in data), recoding (which refers to operations such as binning and data type transformation) and clean up (which refers to deleting columns with missing values to create a "clean" dataset) (Williams, 2011g). In the realm of model building it provides kmeans , clara, hierarchical as models for clustering tasks (unsupervised learning) (Williams, 2011b) (see fig. 3.2). It supports data discovery tasks such as associations analysis, and Supervised analysis with decision trees/Forests (see fig. 3.1) and SVM's (Williams, 2011c, 2011e, 2011f), and boosting (Williams, 2011a). And as for the evaluation methods, it supports supervised models evaluations such as precision, confusion matrix or error matrix, ROC curve and so on (Williams, 2011d), it also supports association rules evaluation metrics such as the lift (McNicholas, Murphy, & O'Regan, 2008) (see fig. 3.4). However, it's important to note that Rattle does not support all of R's algorithms nor evaluations such as neural networks for instance (Bergmeir, Benítez Sánchez, et al., 2012).

TW_8 **Takeaway 8.** *Developing machine learning workflows using R code can be complex for certain users. It was necessary to have support for guiding workflow selection and visualization provided through tools like Rattle.*

3.1.2 Weka (Waikato Environment for Knowledge Analysis)

Weka (Hall et al., 2009), a Java-based open-source data mining (DM) platform, originates from the University of Waikato in New Zealand. Released under the GNU GPL 3 for non-commercial use, Weka has maintained a stable popularity over the years. Its appeal primarily lies in its user-friendly interface and a wide array of implemented DM algorithms (Arora, 2012).

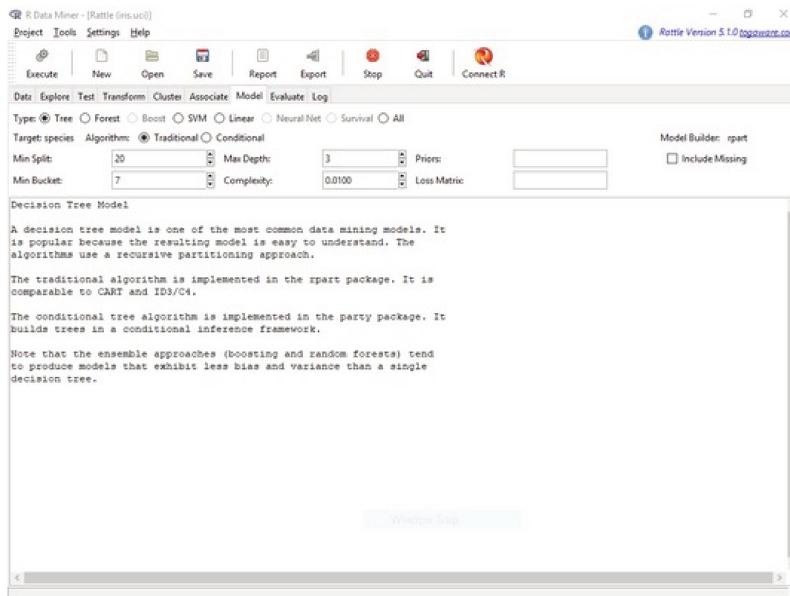


Figure 3.1: The rattle classification tab

In this tab, the user can choose the type of supervised algorithm to utilize and set its parameters. Options include Decision Tree, Random Forest, Support Vector Machine (SVM), Linear Regression, Neural Networks, or the ability to compare all of them simultaneously

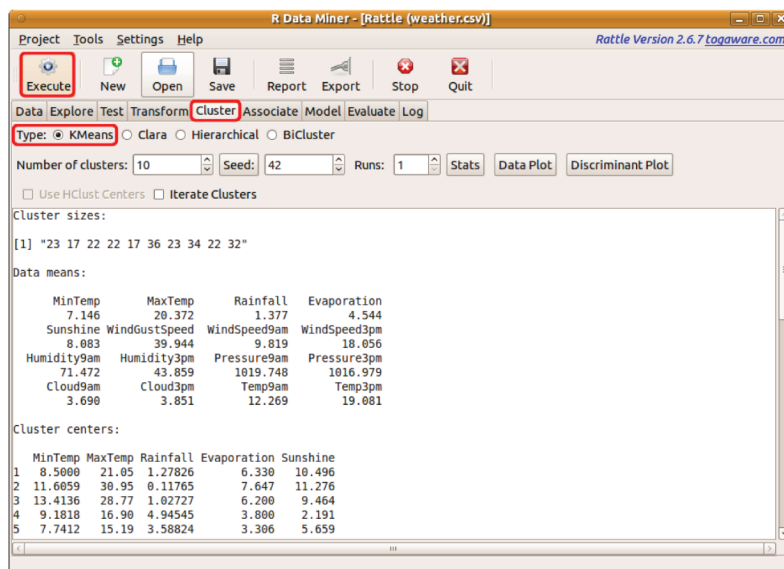


Figure 3.2: Rattle clustering tab

In this tab, the user can select the type of unsupervised algorithm to use and set its parameters. The options include K-Means, Hierarchical Clustering, CLARA, Bicluster, or the option to compare all of them simultaneously.

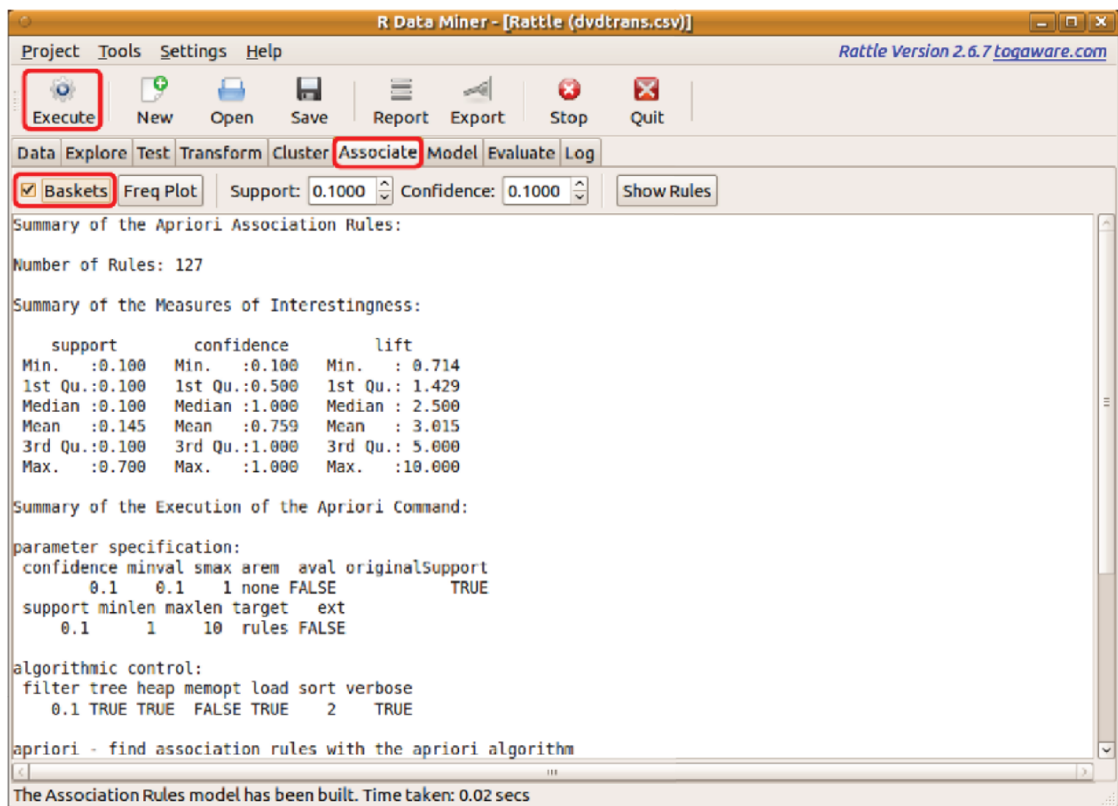


Figure 3.3: Rattle association rules tab

In this tab, the user can configure the parameters for the association rules algorithm, including setting values for support and confidence.

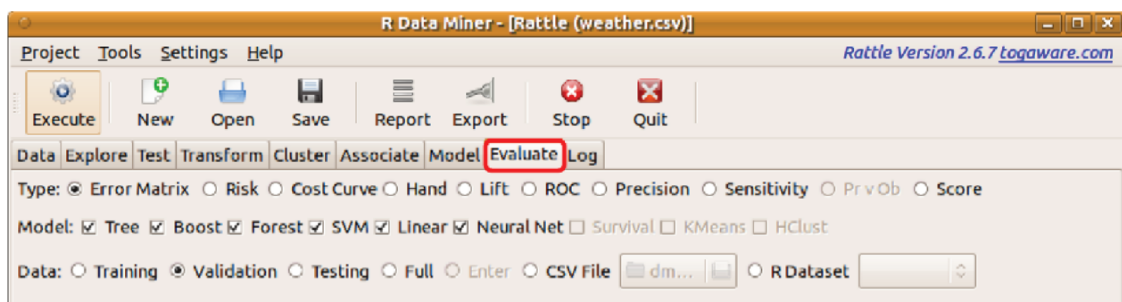
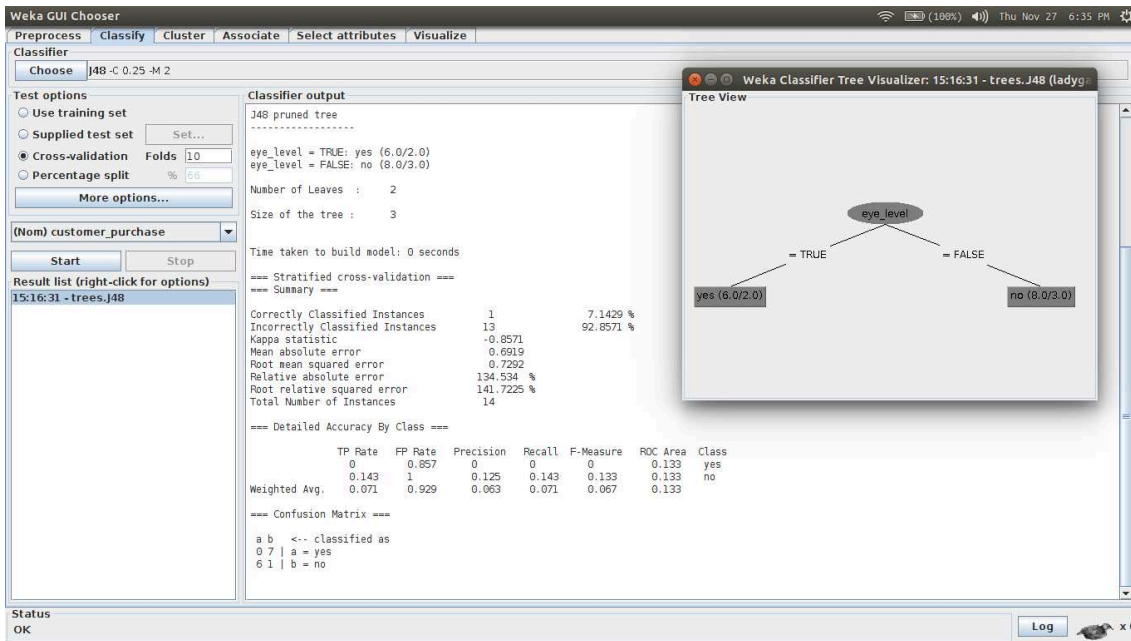


Figure 3.4: Rattle evaluation tab

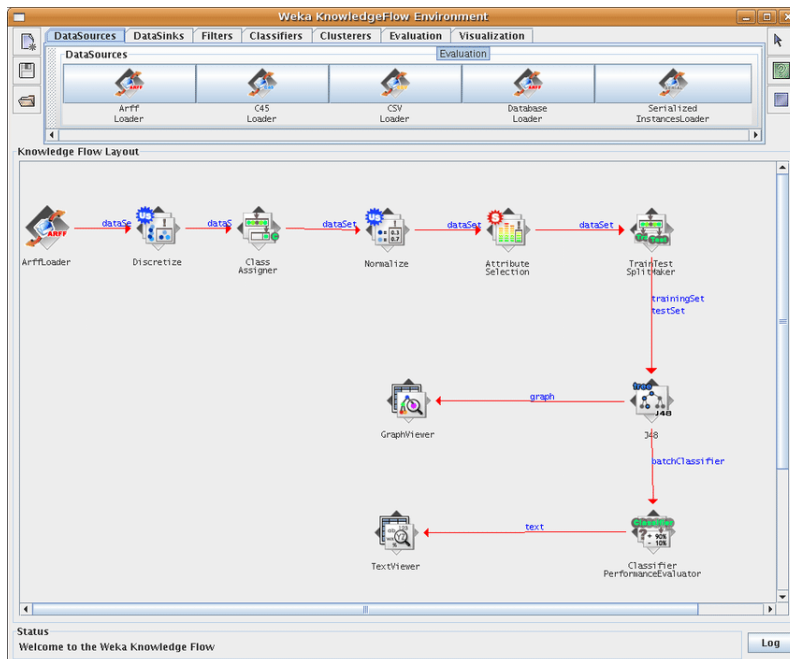
In this tab, users can select the evaluation type and the model they want to assess. Choosing "Error Matrix" will result in the automatic deselection of models such as Survival, Kmeans, and Hclust because they are not suitable for supervised evaluation metrics.

Furthermore, users can specify the data split for the evaluation, including options for training, validation, testing, or using the entire dataset.



fig_weka

Figure 3.5: The weka decision tree model tab
 In this tab, users can choose a classifier, define data splits for training, and visualize the results. The history of previous results is stored in the results list located at the bottom left of the interface.



fig_wekaKF

Figure 3.6: Weka Knowledge flow layout example
 In this tab, users can view and interact with the workflow layout.

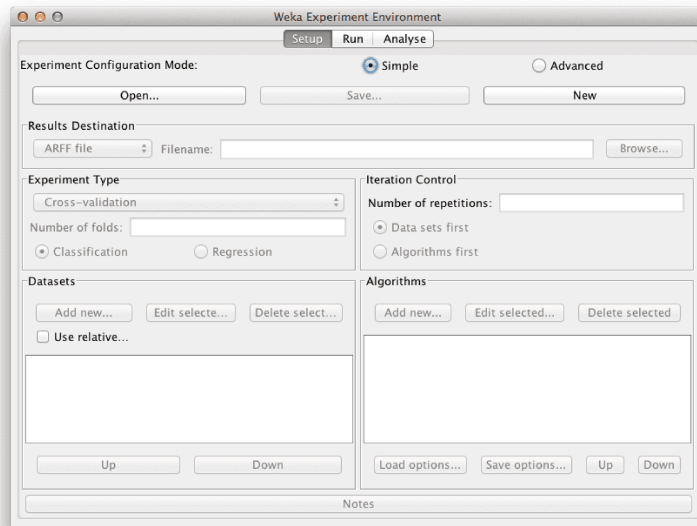


fig:wekaExp

Figure 3.7: Weka Experimenter tab

In this tab the user designs the experimental setup.

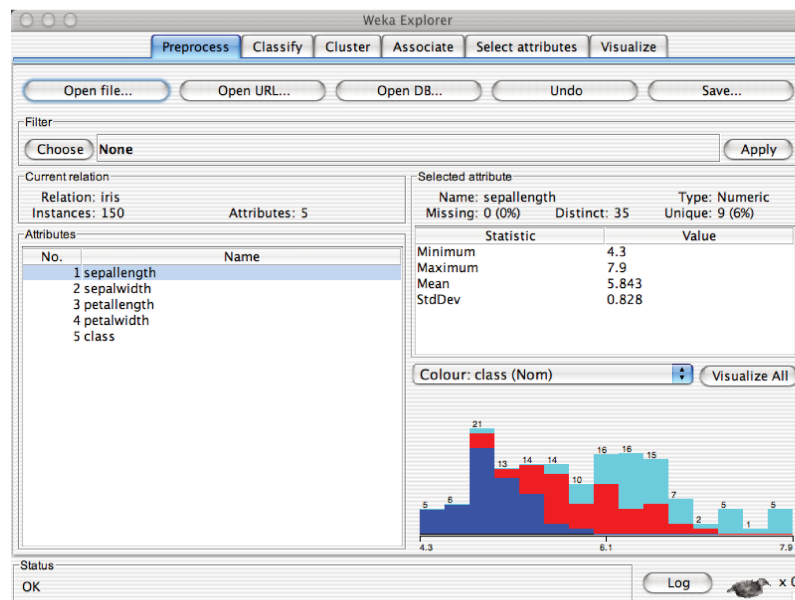


fig:wekaExplorer

Figure 3.8: Weka explorer tab

In this tab, users can explore the data and visualize the predefined analysis performed by Weka on the input datasets.

Weka offers users four distinct options for conducting DM tasks: a command-line interface (CLI), the Explorer, the Experimenter, and Knowledge flow. The Explorer, being the preferred choice, facilitates tasks such as defining data sources, data preparation, machine learning algorithms, and data visualization (see fig. 3.8). The Experimenter, on the other hand, is primarily used for comparing the performance of different algorithms on the same dataset (see fig. 3.7). Meanwhile, Knowledge flow follows a paradigm similar to RapidMiner's operator concept, allowing users to specify data flows using interconnected visual components (see fig. 3.6).

Weka also offers support for various supervised models evaluation procedures and metrics (Gnanambal, Thangaraj, Meenatchi, & Gayathri, 2018). However, it does have limitations in terms of data survey and visualization methods. It is more inclined toward addressing classification and regression problems than descriptive statistics and clustering. It's important to note that Weka's support for big data, text mining, and semi-supervised learning is somewhat limited (Kotak & Modi, 2020), and it is currently introducing neural networks methods such as Auto Encoders for anomaly detection (Macías-García et al., 2017), Recurrent neural networks to deal with time series data prediction (HaddadPajouh, Dehghantanha, Khayami, & Choo, 2018), Convolution neural networks for image classification (Zainudin, Shamsuddin, & Hasan, 2019) to address a key limitation for the tools adoption.

TW_9 **Takeaway 9.** *Weka incorporates essential safety features to assist users in selecting their workflows. For example, it restricts the evaluation of a supervised algorithm with association rules evaluation metrics, preventing scenarios that could occur outside of Weka, such as in R.*

3.1.3 Knime (Konstanz Information Miner)

KNIME (Konstanz Information Miner), a general-purpose data mining (DM) tool, is based on the Eclipse platform and developed and maintained by the Swiss company KNIME AG (Berthold et al., 2009). KNIME is utilized by over 1920 organizations in more than 60 countries, demonstrating substantial community support.

KNIME adopts the visual programming paradigm enabling users to construct workflows by placing and connecting visual building blocks known as nodes (Fillbrunn et al., 2017) (see workflow editor in fig. 3.9 and fig. 3.10). And similarly to weka it provides an Explorer (see knime explorer in fig. 3.9). KNIME offers more than 1000 nodes through its core installation and extensions, facilitating diverse data processing tasks. These nodes are organized hierarchically, searchable by name through an intuitive interface, and come with detailed documentation, automatically displayed upon selection (see "Node repository in the bottom left and node description in the right part of fig. 3.9).

One of KNIME's most significant strengths lies in its seamless integration with Weka and R (Dwivedi, Kasliwal, & Soni, 2016). This integration, while requiring extension installation, brings forth an abundance of functionalities from Weka and allows the incorporation of R code within workflows.

TW_10 **Takeaway 10.** *There is a need for a comprehensive framework to facilitate the complex process of composing machine learning (ML) workflows, particularly for beginners faced with numerous choices. It has led to the development of frameworks and user interfaces on top of classical libraries to provide a structured approach that supports streamlining the decision-making process and making it more accessible for practitioners looking to build ML solutions.*

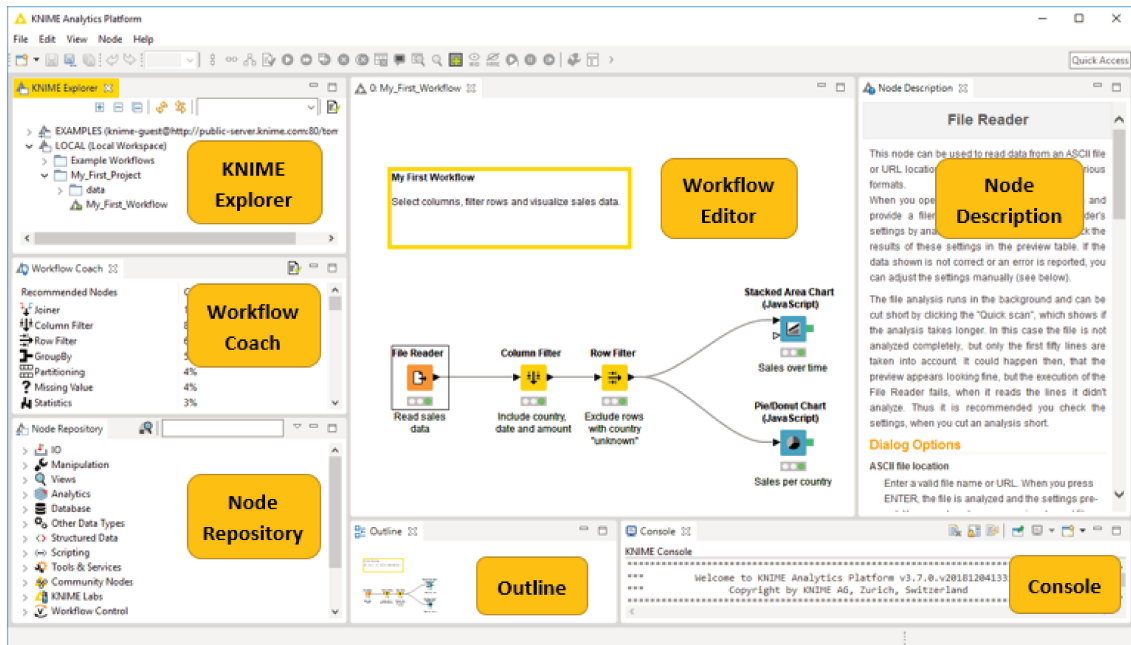


Figure 3.9: Knime workbench tab

fig_knime2

This is an overview of the KNIME Workbench, highlighting its main components: the Explorer, Workflow Editor, Workflow Coach, Node Repository, Workflow Outline, and Console.

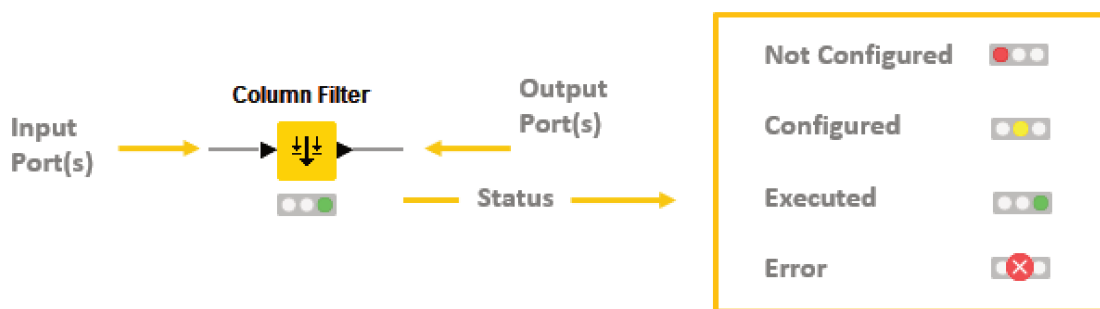


Figure 3.10: Knime Node ports and node status

fig_knime3

While constructing a workflow using the Workflow Editor in KNIME, the software provides real-time status updates for each node, indicating whether it is configured or not. During execution, it also informs you which nodes can be executed successfully and which ones encounter errors.

3.2 Sharing data science experiments

`ss_reuse`

3.2.1 Reuse of notebooks by clone-and-own

In a data science case study, small changes in how data is collected, annotated, cleaned, or processed may lead to different results. For this reason, data scientists must document their data analysis and processing stages. This documentation step is even more crucial if someone else has to understand, maintain or trust the initial work (Rule, Tabard, & Hollan, 2018). Notebooks have clear benefits, in such a context, by providing storytelling through cells of code intertwined with in-line documentation. The data scientists have thus widely adopted computational notebooks, even at the heart of the internal data analysis infrastructures of companies such as Netflix (Matthew, Kyle, & Michelle, 2018) or IBM (A. X. Zhang et al., 2020).

The recipe for this success is a clever combination of code, visualization, and textual documentation, all in a single document. The fact that these notebooks and the associated tools and technologies can also be open source, with Jupyter Notebook as the most famous one, makes them the number one choice for anyone who wants to explore data and share their findings (Perkel, 2018). However, the freedom offered by notebooks and the iterative and exploratory nature of building ML workflows affect their reusability. Among the reasons, one can find a need for more documentation on the workflow (Kery, Radensky, Arya, John, & Myers, 2018; A. Y. Wang et al., 2021), the difficulty of replicating the experiment (J. Wang, Kuo, Li, & Zeller, 2020) and the diversity of problems. To tackle this, data scientists only clone the relevant parts and use them in their own workflows (Koenzen, Ernst, & Storey, 2020), naturally following the so-called *clone-and-own* practice (Dubinsky et al., 2013; Rubin, Czarnecki, & Chechik, 2013; Kehrer, Thüm, Schultheiß, & Bittner, 2021; Krüger & Berger, 2020). Data scientists find the same advantages to using the clone-and-own approach as in previous empirical studies (Dubinsky et al., 2013). Cloning is simple and directly available for developers that are then independent. The experimental nature of producing solutions in machine learning makes the approach even more compelling. Cloning saves time, especially by helping to prepare data and fix parameters. Therefore, the ability to clone a notebook is crucial, as the variability of problems means that it is rare to be able to reuse a notebook without modification for a new data set or when the requirements of the problem change. The autonomy provided by cloning becomes essential in ensuring adaptability to new problem domains.

Despite the advantages of the autonomy offered by cloning, managing the integration of cloned artifacts, such as cells or code segments, can be challenging, as it can result in an inappropriate combination of algorithms and impede the reuse process.

It remains challenging to construct appropriate workflows (Amershi et al., 2019) because the interactions between the current data, the composition of algorithms, and the business requirements are substantial and not always well understood. Sculley et al. summarise these interactions as follows: "*changing anything, changes everything*" (Sculley et al., 2015).

Data scientists adapt cloned sources, by preparing the data to match the expectations of the solution captured by the initial notebook, or by eliminating unnecessary processing. These notebook adaptations are common, but can impact the entire model-building process. Data scientists also choose which notebooks to clone based on criteria such as origin, readability, and quality of results conforming to "*Reuse occurs through personal knowledge, memory, and networking*" (Dubinsky et al., 2013). In a non-classical case, determining which notebook to use as a source for cloning is probably the most important issue. To establish a match between the solution proposed by

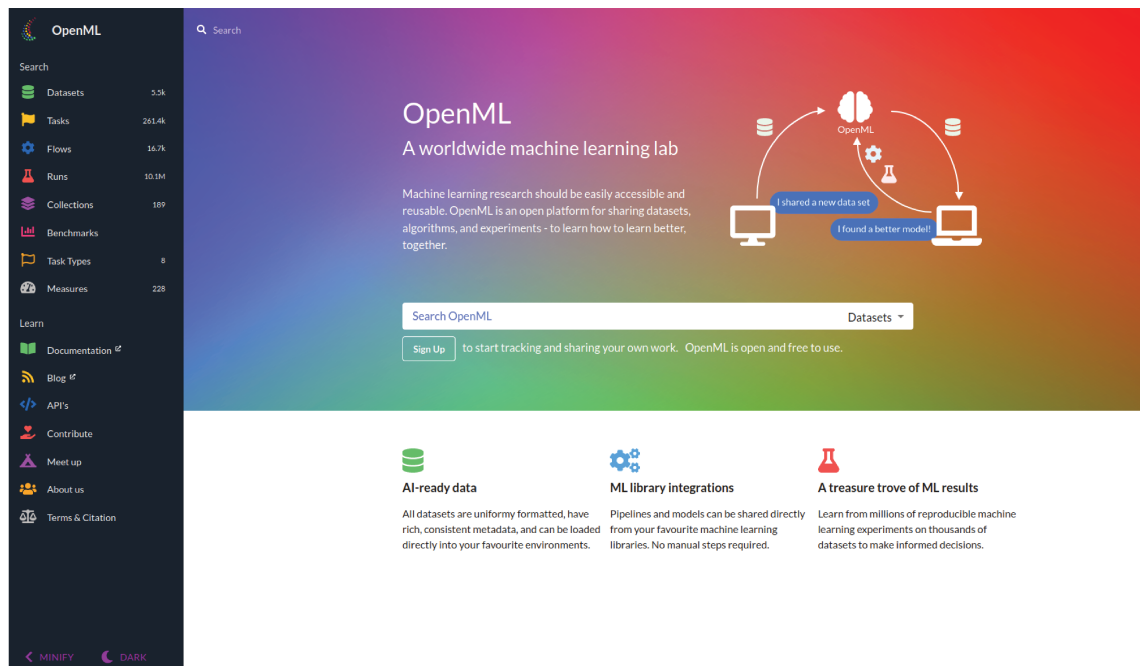


Figure 3.11: OpenML home page

fig_openML1

the source notebook and the new problem, data scientists must consider many parameters beyond metadata or targeted learning. If we make an analogy, it is not enough to know that we want to *sort* to choose a data structure; we need to know its usage, the size of the data, the evolution of the data and the target of the deployment.

TW_11

Takeaway 11. *The development of computational notebooks by clone-and-own is a natural, necessary, and at the same time, difficult approach (Koenzen et al., 2020) involving heavy, error-prone, and particularly uncertain activities.*

3.2.2 OpenML

OpenML serves as a collaborative platform for data science experiments, fostering efficient sharing and collaboration through its core components, including datasets, tasks, flows, and runs (Van Rijn et al., 2013) (see fig. 3.11). Datasets provide a foundational resource for researchers, enabling the execution of various experiments. Tasks define specific objectives and protocols for these experiments, ensuring a clear focus on shared scientific challenges. Flows encapsulate the machine learning workflows or algorithms used to tackle tasks, making it possible for users to contribute and replicate implementations and methodologies. Runs represent the tangible results of these experiments, including predictions, models, and evaluations, serving as a crucial avenue for sharing findings within the data science community.

In addition to these core components, OpenML extends its utility through plugins that enable compatibility with external tools. OpenML seamlessly integrates with Weka (Van Rijn et al., 2013), R (Vanschoren, Van Rijn, Bischl, & Torgo, 2014), RapidMiner (Van Rijn & Vanschoren, 2015), and KNIME (Vanschoren, Rijn, & Bischl, 2015), allowing users of these popular data science and machine learning software to access OpenML datasets, tasks, and results directly

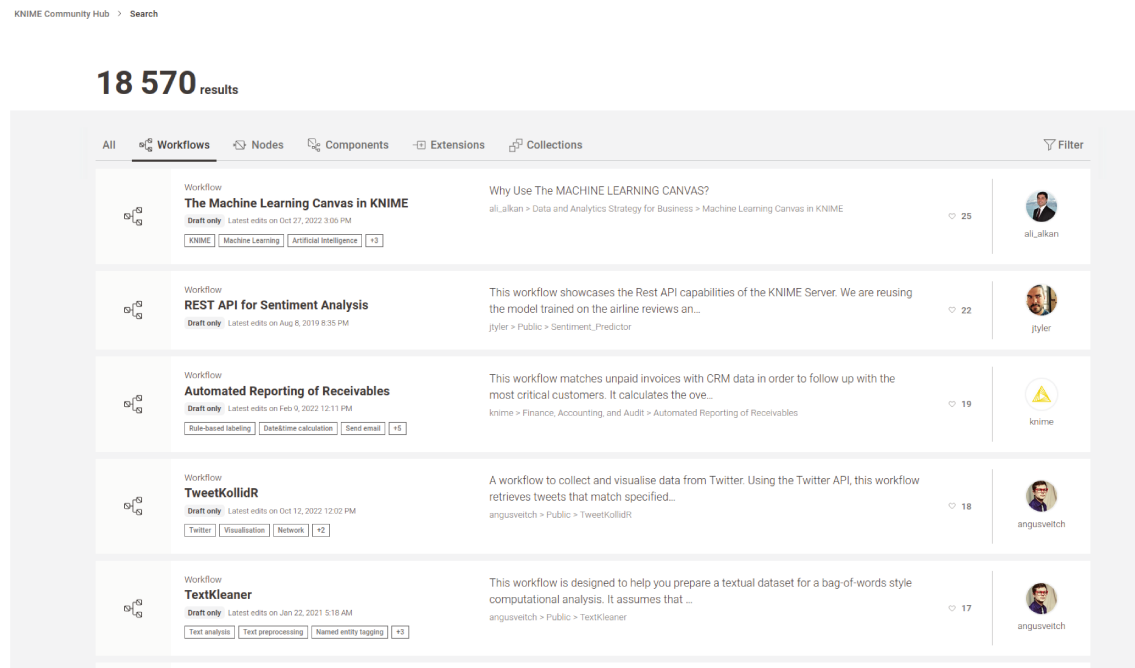


Figure 3.12: Knime hub workflows repository

In this hub, users can share, retrieve, and collaborate on machine learning workflows, nodes, components, and extensions.

within their familiar environments. This enhanced accessibility and integration further promote OpenML's role as a collaborative hub, facilitating the sharing and replication of data science experiments across various platforms and tools.

Takeaway 12. *The machine learning community highly values the sharing of machine learning experiments from datasets to machine learning workflows. The OpenML platform facilitates this by automating metadata analysis and enabling comparisons between different datasets and their associated workflows. It's important to note that this analysis primarily focuses on data and does not incorporate stakeholder requirements which scopes the ML problem one of its two dimensions only.*

3.2.3 KNIME Hub

The KNIME Hub is a central hub managed by KNIME, functioning as a repository and collaboration space for the KNIME Community (Ordenes & Silipo, 2021). It offers powerful search and collaboration features, enabling users to discover and access a variety of shared resources.

It facilitates the sharing of nodes, extensions, components, and workflows. Users can contribute and access these resources, which can serve as building blocks for data science solutions within the KNIME Analytics Platform (Thomas & Gehrig, 2020). Additionally, users can upload, organize, and collaborate on workflows.

Users can efficiently retrieve experiments and resources from the KNIME Hub using various features. The search bar allows users to locate specific items shared by the KNIME Community. Results can be further filtered by item type, and clicking on an item provides detailed information

about its functionality, owner, and related workflows or nodes. Users can easily incorporate these resources into their workflows through a convenient drag-and-drop feature. Furthermore, users can save valuable items by liking them, and engage in discussions related to specific workflows. Additionally, short links can be generated for sharing specific items with others. KNIME Hub serves as a collaborative platform for sharing and accessing data science resources within the KNIME Community.

TW_13

Takeaway 13. *Users can access machine learning workflows, nodes, and extensions by searching for keywords on the KNIME Hub. Each artifact is accompanied by comprehensive documentation, including information about the owner and its functionalities. This feature allows users to select their preferred artifacts, reflecting the importance of familiarity in the choice of artifacts for data scientists.*

Furthermore, the platform supports discussions on workflows, emphasizing the need for post-mortem analysis of workflows and encouraging interactions and discussions among data scientists.

3.3 Synthesis

The diversity of solutions arises from the variability throughout the entire process, including algorithm selection, data preparation, and workflow design. This diversity is further amplified by the individual modeling decisions made by data scientists, leading to distinct results.

Therefore there is a need to quickly eliminate inadequate solutions and mutualize solutions based on varying problem contexts.

CHAPTER 4

Software product lines

'''

4.1	Introduction	37
4.2	Feature models	37
4.3	Lopez-Herrejon and Batory standard example	38
4.4	SPL's for composing scientific workflows	39
4.5	Feature models edition	40
4.6	Evolution-aware evaluation	41
4.7	Synthesis	44

4.1 Introduction

Software Product Line Engineering is concerned with the identification and representation of shared characteristics, common attributes, and universal features among a group of software-intensive applications within a specific problem domain (Pohl, Böckle, & van Der Linden, 2005). This approach enables the swift development of variations of domain-specific applications by configuring a set of reusable assets, known as core assets. These core assets help manage both common elements and variations. Commonality is addressed by allowing domain analysts to capture shared conceptual information across applications within the same domain, while variability is handled by enabling domain analysts to incorporate application-specific attributes and features into a unified model.

In the context of software product lines, one of the significant techniques for domain modeling is feature modeling (Czarnecki, Helsén, & Eisenecker, 2005) because it can capture and model variability within software product lines (Babar, Chen, & Shull, 2010).

4.2 Feature models

Features represent distinguishing aspects, qualities, or characteristics of a group of systems (K. Lee, Kang, & Lee, 2002). They help depict the shared structure and behavior among similar systems. To create a product line, these features from similar or related systems are organized into a feature model. A feature model represents the potential configuration space of all products within a system product line based on their features. Therefore, it is crucial for feature models to capture both variability and commonality among features across different applications within a given domain. In essence, feature modeling is an approach for domain modeling that captures both the variability and commonality present in the features of applications within a specific target domain (Bagheri, Ensan, & Gasevic, 2012).

Feature models provide a practical way to model commonality by allowing domain modelers to create a shared feature model representation for multiple applications. Additionally, they capture variability by providing the means to model competing features from different applications under a unified umbrella. For example, commonality is represented when a similar feature, existing in multiple applications, is portrayed as a unique feature in the overall domain representation. Conversely, variability is exemplified when separate applications view a concept differently and are therefore modeled using competing features.

In its graphical representation a feature model typically consists of a tree structure with the root node representing a domain concept (e.g., a domain application), while other nodes and leaves represent various features. Therefore, a feature serves the function of either pointing out similarities or setting apart variations in a product line by addressing **user-visible** functional or non-functional needs (Tessier, Gérard, Terrier, & Geib, 2005).

fig_mandatory



Figure 4.1: Mandatory feature group

Within feature models, features are hierarchically organized and can be classified into several categories:

ig_optional



Figure 4.2: Optional feature group

alternative

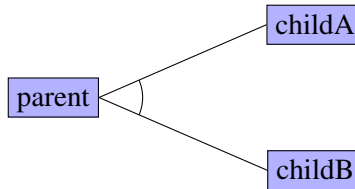


Figure 4.3: Alternative feature group

Mandatory features, which must be included in the description of their parent feature, used to represent commonality (see fig. 4.1).

Optional features, which may or may not be included in their parent description, depending on the situation (see fig. 4.2).

Alternative feature groups, where only one feature from the group can be included in the parent description (see fig. 4.3).

Or feature groups, where one or more features from the group can be included in the description of the parent feature (see figure 4.4).

Additional constraints, referred to as Integrity Constraints (IC), are often added to feature models to capture mutual interdependencies among features. These constraints come in two types: "Includes" constraints (equivalent to a logical " $A \implies B$ "), indicating that the presence of a given feature (or set of features) requires the existence of another feature (or set of features), and "Excludes" constraints (equivalent to a logical " $A \implies \neg B$ "), signifying that the presence of a given feature (or set of features) necessitates the elimination of another feature (or set of features).

4.3 Lopez2001standard Lopez-Herrejon and Batory standard example

FMsexample

To further illustrate feature modeling concepts, let's explore the Graph Product Line (GPL) (Lopez-Herrejon and Batory, 2001), which is designed to serve as a benchmark for evaluating feature modeling techniques (see fig. 4.5). GPL is intended to enable the creation of configurations that address various problems in the domain of graph manipulation. For example, GPL can be configured to execute different graph search algorithms on directed or undirected graph structures.

fig_or

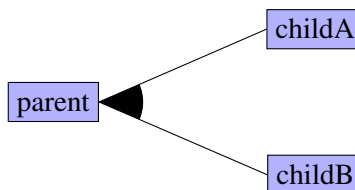
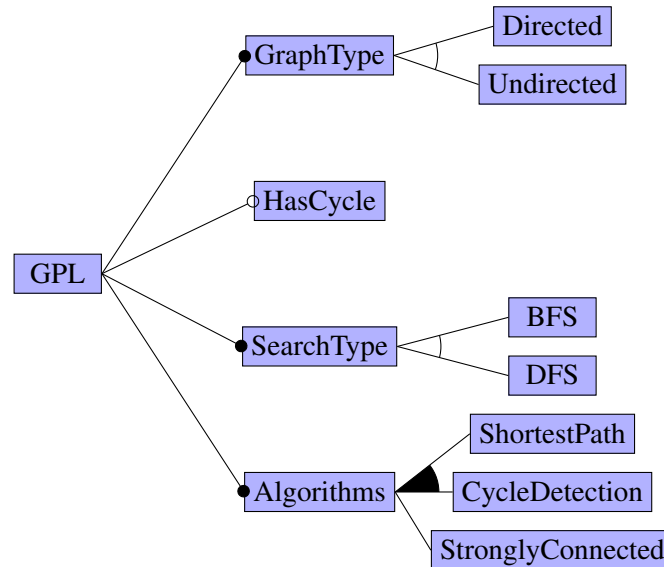


Figure 4.4: Or feature group

As shown, GPL consists of three primary features:



Cross-tree constraints

$HasCycle \implies \neg CycleDetection$

$CycleDetection \implies \neg BFS$

$CycleDetection \implies DFS$

fig_GPL

Figure 4.5: GPL feature model

Graph Type: Features that define the structural representation of a graph.

HasCycle : Feature that defines the presence of cycles in the graph

Search: Traversal algorithms represented as features that facilitate graph navigation.

Algorithms: Additional useful algorithms for manipulating or analyzing a given graph.

It’s important to note that not all possible feature configurations within GPL result in valid graph programs. For instance, a configuration of GPL designed to check if a graph is strongly connected cannot be implemented on an undirected graph structure. These restriction rules are Integrity constraints. Here are some examples :

4.4 SPL’s for composing scientific workflows

Scientific workflows, characterized by structured sequences of tasks, often automated, are pivotal in processing data, conducting analyses, and producing results (Georgakopoulos, Hornick, & Sheth, 1995).

Consider a medical researcher investigating Alzheimer’s disease as presented in [Lorenzi, Ayache, Frisoni, Pennec, et al.](#). The task involves tracking brain atrophy over time by analyzing MRI images at various stages in a patient’s life. The complexity lies not in comprehending scientific

concepts but in the arduous manual process of selecting and configuring tools. Without assistance, designing these workflows becomes a labor-intensive and error-prone effort, hampering scientific progress (McPhillips, Bowers, Zinn, & Ludäscher, 2009; Gil et al., 2007).

Practitioners therefore need assistance in designing scientific workflows, which SPLs provide by mutualizing the strengths of individual software components into a unified suite of products, and streamlining workflow creation (Camillieri et al., 2016; Svahnberg & Bosch, 1999).

Expanding upon the example presented in (Lorenzi et al., 2010), Acher et al. introduce distinct brain image analysis workflows formalized as separate Software Product Lines (SPLs), referred to as "services." These services are subsequently amalgamated into a higher-level SPL, termed a family of services. These families of services are then further integrated into a catalog of services, where all scientific workflows can be comprehensively represented. To make this integration possible, they proposed a tool-supported approach that supports the merging of features, checks consistency in feature model constraints, and checks the validity of product configurations. Importantly, the approach maintains records of all the merged SPLs because of the loss of information, such as the removal of conflicting constraints from distinct SPLs during the composition process.

TW_14 **Takeaway 14.** *Integrating various components, each with its unique constraints, into a cohesive suite of software product lines for scientific workflow development simplifies the overall process. However, it necessitates a stable and structured organization of individual components, each addressing their specific concerns, before the composition phase. Additionally, as composition may result in the loss of unspecified constraints, maintaining a record of the products is essential for post-mortem analysis and reasoning about the inferences made by the software product lines.*

4.5 Feature models edition

Modifying a feature model results in the creation of a new model, and the primary objective is to understand how these modifications impact the product line of the original model. These modifications can be categorized into distinct types: refactoring, which involves making changes without introducing or removing products; specialization, where existing products are removed without the introduction of new ones; generalization, which entails adding new products without removing existing ones; and arbitrary edits, representing changes that do not neatly fit into these predefined categories.

In addition to these fundamental operations, Acher, Collet, Lahire, and France. introduced two composite operations. The insert operation is intended for the insertion of a feature model or a feature beneath a specified feature within the target feature model, taking into account an FM operator such as XOR or groups. Conversely, the merge operation aims to consolidate shared aspects from multiple feature models. The merge operation is presented in two formats: the union merge, where the resulting feature model includes all configurations valid in either of the input feature models, and the intersection merge, where the output feature model includes configurations valid in both input models.

TW_15 **Takeaway 15.** *Systematic feature model editing relies on identifying and utilizing relevant patterns, such as merge and insert patterns. It's crucial to design systematic editing patterns that align with potential evolutions within the product line.*

4.6 Evolution-aware evaluation

In this section, we delve into the quantitative assessment of software product lines using a key set of metrics from the existing literature. Subsequently, we will explore the existing correlations between some of these quantitative metrics and the qualitative assessment related to SPLs' maintainability (Bagheri & Gasevic, 2011).

With these aspects in mind, we will present a subset of variability model metrics, directing interested readers to the literature review by El-Sharkawy, Yamagishi-Eichler, and Schmid for a more comprehensive overview. These variability model metrics fall into four categories: modifiers, size metrics, ratio and complexity metrics, and element metrics (El-Sharkawy et al., 2019a).

Modifiers : These modifiers provide alternative perspectives on the variability model, opening new avenues for observation and analysis.

Typed features and attributes introduce the possibility of associating non-Boolean attributes with features, such as strings or numbers. Additionally, they categorize features as either configurable or constant, based on their configurability and their relation to other features in the hierarchy (KANG, COHEN, HESS, NOVAK, & PETERSON, 1990; Berger & Guo, 2014).

Atomic sets simplify the model by consolidating connected mandatory features into atomic sets. This reduction in complexity is particularly beneficial for automated verification and preserves the original model's variability. Moreover, different types of influence, including positive ($A \implies B$), negative ($A \implies \neg B$), preconditional influence (A requires B), and positive or negative influence (none of the above categories apply), can be employed to describe dependencies between atomic sets (W. Zhang, Zhao, & Mei, 2004).

Variability points complement atomic sets by defining the number of decisions necessary to configure a feature diagram. This encompasses optional features, feature alternatives, and OR-groups (Mann & Rock, 2011; Berger & Guo, 2014).

Type of constraints can be categorized based on their complexity and purpose (Passos et al., 2011). Constraints fall into three primary categories: pure Boolean, pure non-Boolean, or mixed constraints. This classification enhances our understanding of the model's complexity and the role of constraints, whether they function as "existence" conditions, "value/range" restrictions, or "default" values (El-Sharkawy et al., 2019a).

Size Metrics: Several size metrics are employed to assess variability models. The most prominent among these is the "Number of Features" metric, which comes with various derivations (C. I. Bezerra, Andrade, & Monteiro, 2014; Berger, She, Lotufo, Wasowski, & Czarnecki, 2013; Berger & Guo, 2014; Passos et al., 2011; Vyas & Sharma, 2016). Additional size metrics evaluate the number of constraints, the maximum depth of the tree structure (C. I. Bezerra et al., 2014; Berger et al., 2013; Mefteh, Bouassida, & Ben-Abdallah, 2015; Vyas & Sharma, 2016) and the configuration space's size, representing the total count of valid configurations.

The following metrics are derived from "Number of features":

Number of Features (NoF): This measures the total count of features in the feature model.

Number of Leafs: It quantifies the number of features with no children or specializations (Maâzoun, Bouassida, & Ben-Abdallah, 2016; Vyas & Sharma, 2016).

Number of Constraints Features: This metric calculates the number of features involved in cross-tree constraints (C. I. M. Bezerra, Monteiro, Andrade, & Rocha, 2016; Sánchez, Segura, & Ruiz-Cortés, 2014).

Number of Atomic Sets: It counts the total number of atomic sets (Mann & Rock, 2011).

Number of Variability Points: This metric quantifies the number of variability points (Mann & Rock, 2011).

Number of Dead Features: It represents the number of features that cannot be selected without violating constraints. For instance, if feature A is mandatory and a constraint (C) enforces $A \implies \neg B$, then B is never selectable without violating constraint (C) (Mann & Rock, 2011).

The following metrics are derived from constraints measurement:

Number of Constraints (NoC): This metric encompasses the total number of cross-tree constraints (excluding non-propositional constraints enforced by the feature model structure, such as alternative/OR-group relationships) (Passos et al., 2011).

Number of OR Feature Groups: It assesses the total number of OR feature groups (Berger & Guo, 2014; C. I. M. Bezerra et al., 2016; Leitner, Weiß, & Kreiner, 2012).

Number of XOR Feature Groups: This metric evaluates the number of alternative feature groups (XOR feature groups) (Berger & Guo, 2014; C. I. M. Bezerra et al., 2016).

Interface Complexity: This metric considers OR/XOR feature groups as cross-tree constraints and is therefore the sum of the previous three metrics (Leitner et al., 2012).

Additionally, tree depth (DT) measures the longest path from the root feature to leaf features, and there are derivations that calculate the mean or median tree depth (C. I. Bezerra et al., 2014; Mefteh et al., 2015). Finally, the "Number of Valid Configurations" refers to the count of valid products, complying with cross-tree constraints, that can be generated with a given feature model.

Ratio and complexity metrics: aim to convey the intricacy of the examined variability model. They often result from a combination of multiple size metrics.

The following metrics are examples of such ratios:

Cross tree-constraints ratio (CTCR) is a ratio calculated by dividing the number of distinct features involved in the cross-tree constraints by the total number of features in the FM (Sánchez et al., 2014; Vyas & Sharma, 2016)

Cyclomatic complexity of feature models (CC) counts the number of distinct cycles in a feature model. And because a cycle in feature models can only be generated by the cross-tree constraints. This metric could be simplified to the number of cross-tree constraints (Bagheri & Gasevic, 2011; Mefteh et al., 2015; Vyas & Sharma, 2016).

Ratio of variability (RoV)/ Branching factor this metric calculates ratio between the average number of child features divided by the average number of parents features (Bagheri & Gasevic, 2011; Berger et al., 2013; C. I. M. Bezerra et al., 2016).

Flexibility of Configuration (FoC) metric is calculated as the ratio of the number of optional features to the total number of available features in the feature model (C. I. Bezerra et al., 2014; Vyas & Sharma, 2016; C. I. M. Bezerra et al., 2016). The underlying idea is that when there are more optional features in the feature model, it provides designers with a greater range of choices during the configuration process. In other words, FoC quantifies the level of flexibility in configuring the feature model.

Elements metrics are measures designed for individual elements within a variability model.

In the following we present prevalent element wise assessment metrics:

Commonality of feature (Comm) assesses the reuse ratio of a feature, labeled as "f," within a Software Product Line (SPL). It is calculated by dividing the count of products where feature "f" is included by the total number of products in the SPL (Sánchez et al., 2014).

Theoretical selection ratio of feature (feature coverage) calculates the percentage of configurations in which each feature is selected (Mann & Rock, 2011).

Theoretical deselection ratio: calculates the percentage of configurations in which each feature is deselected (Mann & Rock, 2011)

Metrics for maintainability: In their study (Bagheri & Gasevic, 2011), the authors investigated the correlations between size metrics and qualitative aspects related to:

Analyzability: The extent to which the conceptual model of a software system can be evaluated for shortcomings.

Changeability: Involves the potential and ease of making modifications to the model as needed.

Understandability: Refers to the likelihood that the software system's model can be understood by its users and other model designers.

Their findings revealed several correlations. For example, the number of leaf features is negatively correlated with analyzability, changeability, and understandability. In other words, the more leaf features the feature model contains the more challenging it is to maintain it.

Similarly, the number of Cross-Tree Constraints (CTCR) and the Flexibility of Configuration (FoC) are negatively correlated with understandability and do not correlate with analyzability and changeability.

This analysis underscores the importance of controlling the growth and complexity of the feature model to maintain its manageability.

TW_16

Takeaway 16. *Metrics play an essential role in monitoring Software Product Lines (SPLs). Various studies have defined metrics tailored to specific needs, enabling practitioners to observe and assess SPLs effectively. However, redundancy exists in the state of the art. For example, metrics*

like Cyclomatic Complexity (CC) and the number of constraints could be seen as closely related, but authors often formulate them differently to address their specific needs.

In retrospect, analyzing these metrics alongside qualitative input from practitioners has revealed correlations that link quantitative metrics to their roles in diagnosing SPLs for consistent evolution. These correlations help bridge the gap between quantitative assessments and qualitative perceptions, enhancing the understanding of an SPL's behavior and its potential for improvement.

4.7 Synthesis

The complexity and diversity of ML problems make it impossible to establish an absolute and definitive framework. The iterative interactions between data scientists and experts to refine solutions result in the emergence of new knowledge. ***Therefore there is a need to adapt to progressive knowledge acquisition.***

Contributions

CHAPTER 5

Capturing Knowledge from trial and error

'''

5.1	Introduction	49
5.2	Experiments design	52
5.2.1	Scenarios description	52
5.2.1.1	Scenario 1	52
5.2.1.2	Scenario 2	53
5.2.1.3	Scenario 3	53
5.2.1.4	Scenario 4	53
5.2.2	Summary table	54
5.2.3	Results description	56
5.2.4	Discussion	58
5.2.5	Explicated knowledge	59
5.3	Conclusion	60

5.1 Introduction

We have seen in chapter 2 some takeaways which are crucial in the design of ML workflows. To exemplify our concepts, let us use an example of anomaly detection in washing machine engines that we will find later in the experiment design as scenario 4. The stakeholders are interested in detecting errors that could potentially cause the machine to break down. In this scenario, the collected data consists of a time series of the engine’s electrical current in volts. The frequency of the engine’s rotation is typically fixed in a normal setup. Experts have annotated abnormal setups when the frequency of rotation deviates from the normal frequency (see an illustration of the normal vs abnormal setup in fig. 5.1. The data is normalized between -1 and 1)

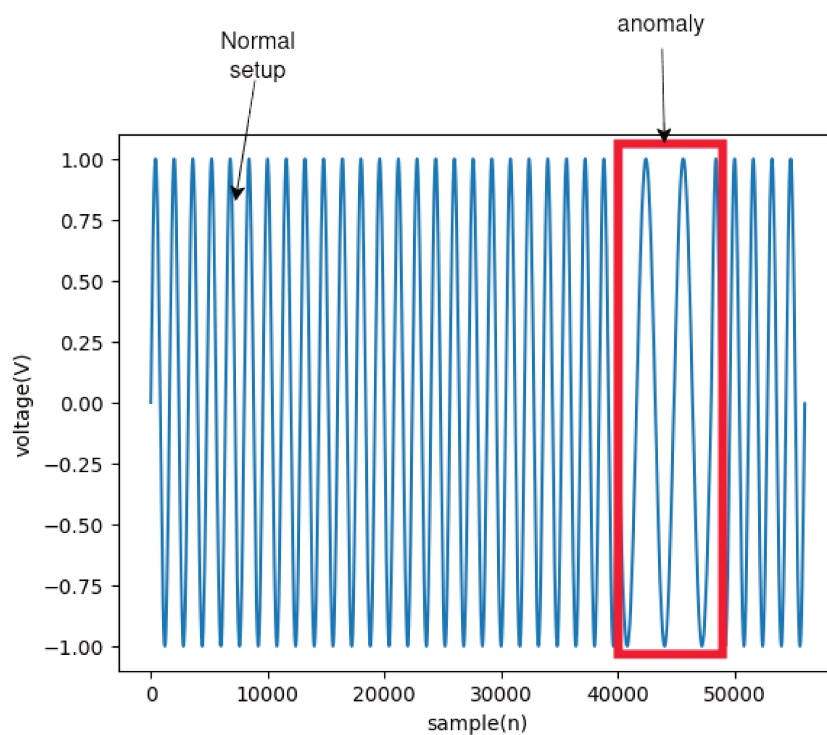


Figure 5.1: washing machine motor engine time series

According to **Takeaway 1** ^{TAW2dimensions} machine learning problems are categorized using data-centric characteristics and business requirements. However, not all data criteria are computable ; some are not present in the training and test sets but are known to domain experts or are encountered in production.

For example, the data scientist cannot determine whether the frequency of anomalies will consistently match the pattern observed in this example or if they may vary in magnitude. This uncertainty persists until the data scientist seeks clarification from the domain expert or encounters different scenarios when deploying the model in production.

As **Takeaway 2** ^{TAW-DSPRACTICES} states, data scientists use a trial-and-error approach and receive domain expert feedback in each iteration to improve their understanding of the problem.

For instance, during the design phase, if the data scientists only encounters time series data with consistent frequency distortions as anomalies, they may overlook the possibility of variations in

anomaly frequency patterns. In this particular scenario, the model trained to detect anomalies will not be considered inaccurate until it encounters such variations in production. However, since the model failure involves not triggering an alarm when an anomaly occurs, the issue may remain undetected for an extended period until the machines break down, a scenario we want to avoid!

In this chapter, we design scenarios to refine our comprehension of the limitations of a specific implementation of the AutoML tool Tpot and automatic hyper-parameter optimization of sklearn models which are standard implementations of models systematically tested by data scientists. The main contribution of this chapter is to illustrate the learning process of the data scientists itself: isolating a component of a machine learning workflow and subjecting it to controlled testing to extract precise insights. These insights can then be shared between data scientists, which will be the subject of the following chapters, *i.e.*, how to represent this information (chapter 6) and how to extend our knowledge (chapter 7).

To illustrate this process, we show how we identified a situation in which Tpot, an AutoML tool, and hyper-parameter optimization of sklearn models (two approaches that require minimal intervention from the data scientist and the 'domain expert) should not be used, for the detection of anomalies in washing machine engines time series. This discovery came from the design of various scenarios to test the approaches, described as a set of criteria detailed in section 5.2.

As a matter of fact, different levels of automation can be applied to machine learning workflows, including fully automated (AutoML) systems that we will call “no-data-scientist-no-domain-expert” or automation level 2 (He, Zhao, & Chu, 2021), systems with automatic hyper-parameter optimization of a set of models selected by a data scientist, that we call “data-scientist-no-domain-expert” or automation level 1 (Luo, 2016), and an approach incorporating domain knowledge and data science expertise that we call “data-scientist-and-domain-expert” or automation level 0 (Brunton, Noack, & Koumoutsakos, 2020).

We study the effectiveness of the three levels of automation for machine learning workflow generation, on standard, modified¹, and generated² time series datasets in a binary classification context to understand their limits.

In section 5.2.1.4, we will exhibit through a non-standard use case of the example presented above how the expert insight can steer the solution in the correct direction while considering data alone produces catastrophic results.

Standard time series datasets for binary classification target well-defined problems and are associated with established evaluation protocols, providing a useful starting point for examining the performance of different approaches. We adapt these standard datasets to align with the reality of industrial applications and discuss how effectiveness evolves with automation levels.

Our results show that automation level 1 and level 2 perform well on standard time series datasets, but their effectiveness decreases as the datasets deviate from the established benchmarks. We also confront these two automation approaches to a workflow resulting from the collaboration of a data scientist and a domain expert (automation level 0) on the example described above of the use-case of time series anomaly detection (Passi & Sengers, 2020; Hoang & Kang, 2019), which involves detecting unusual patterns in time series data. This complex and nuanced task requires a high level of expertise and customization to be effectively solved. Our results show that automation level 0 can effectively solve this task, while automation levels 1 and automation level 2 cannot.

¹We applied modifications to the standard datasets.

²We generated them in accordance with a the real industrial use case Ezako shared with us of the washing machine engine anomaly detection

These findings exhibit essential implications for the practical use of machine learning in real-world settings, where the data for building a machine learning solution is often complex and may not convey the exhaustive criteria that define the problem for instance (*i.e.*, in our example the data at design phase has been collected from one type of washing machines while the solution workflow was expected to work on several types of washing machines therefore the data at design phase didn't convey all properties of the data that is to be met in production.) Additionally, High-performance scores on a given test set can be misleading (Liao, Taori, Raji, & Schmidt, 2021) if the test set does not comprehensively represent the real-world requirements of the use-case (*i.e.*, the test set in our example was from the same type of washing machines as the training set, therefore having excellent performance scores on it can be non representative of the true performances once the solution workflow is confronted to various types of washing machines see results in table 5.2). as a matter of fact when we dealt with this use case in EZAKO the first batch of results was on the training and test set that was first collected from one washing machine and they were stellar using strait forward supervised classification models, however when the model ran on production it raised anomalies only for washing machines of the same type as the training, a domain expert noticed that and conveyed that remark. From that point we re-designed a new semi-supervised machine learning workflow that trains on the same labeled training data but is generalizable to all types of washing machines. It was crucial to design a model that trains on the same training data and not ask for additional labeled data in this context because the domain experts could only afford to spend a small amount of time to label a small additional subset of data that we would use as a hidden test. Particular, complex and nuanced tasks that fall outside 'the beaten paths' require a hands on approach incorporating domain knowledge (Wirth & Hipp, 2000) to solve them effectively . When using such an approach, data scientists can reuse a machine learning workflow that solves a similar task (usually based on their intrinsic perception of the use-case and past use-cases they worked on) or can tailor a machine learning workflow adequately to the needs of the task *i.e.*, In our case since the anomalies showed on the rotation frequency of the engine, we thought it was similar to a previous use case of detecting anomalies on sound frequencies and reused the same workflow, luckily³ this rapprochement saved us trials and errors and pointed us to the right workflow to reuse .

⇒ In accordance with **Takeaway 7** the selected AutoML tool *tpot* does not handle the problem presented in our example and showcased in scenario 4 cf. section 5.2.1.4 as we observe its poor results in cf. section 5.2.3

Therefore, it is crucial for practitioners to carefully consider the specific requirements of the task when deciding which approach to use *i.e.*, in our example, as data scientists we should have inquired about the nature of the collected data with regards to the one in production which falls under business requirements and not assume that the provided data was representative of the one in production without asking the domain expert. Therefore, automation may be a more suitable option for well-defined tasks with established benchmarks. On the other hand, an automation level 0 approach incorporating domain knowledge may be the most appropriate choice otherwise.

This chapter examines time series classification task since it has been identified as a valuable and challenging machine-learning problem (Ismail Fawaz, Forestier, Weber, Idoumghar, & Muller, 2019). We study the effectiveness of the 3 levels of automation aforementioned. Then we extract from the experiments the criteria or deduce the knowledge that could have steered the automation level 0 to the right solution from the first try.

³luckily we noticed the similarity with the sound use case

⇒ *In this chapter we present our approach to refine our understanding of algorithm failure point as we want to avoid wasting time on machine learning workflows bound to fail. We create scenarios in which we move alongside given criteria options and observe the behaviour of the targeted algorithms. in the trial error phase as per **Takeaway 2** (cf. section 5.2).*

5.2 Experiments design

XPdesign

Since time series have not been the first modality of data targeted in this revolution of deep learning, a lot has still to be done and settled to consider that "golden standard models" have been produced as it could be considered with Convolutional Neural Networks (CNNs) for image classification. Time series classification is thus an exciting task to study the three levels of human involvement/automation.

In this section, we will detail scenarios, starting with the context common to the four, then describe the specificities related to each. For our experiment, we showcase the relevance of machine learning the tools on variations of tasks corresponding to use-case scenarios that the user is likely to encounter when dealing with time series classification and when the labels are heavily unbalanced, anomaly detection.

5.2.1 Scenarios description

For all the scenarios, we evaluate automation level 2 by running an AutoML instance based Tpot implementation, with the following hyper-parameters: (i) the training maximum time in minutes `max_time_mins` which we set at 20 minutes for small datasets (with fewer than 300 time series), 40 minutes for medium datasets (with less than 1000 training observations), and 60 minutes for larger datasets (with more than 1000 time series training observations), and 20 minutes for the datasets in scenario 4; (ii) the seed of the random generator `random_state` to 42; and (iii) the score to evaluate the quality of a given workflow generated by Tpot `scoring` to `F1_macro` which is the average F1-score.

For the three first scenarios, we evaluate automation level 1 by running hyper-parameter optimization on the pre-selected models. In these first three scenarios, we have allocated a maximum training time of 5 minutes for small datasets (with fewer than 300 time series training observations), 15 minutes for medium datasets (with less than 1000 training observations), and 30 minutes for larger datasets (with more than 1000 time series training observations). We evaluate all the approaches with the average F1-score on the test set as the performance metric and tracked the confusion matrix for additional information during the experiment.

5.2.1.1 Scenario 1

In the first scenario, we evaluate automation level 1 and automation level 2, in *a supervised paradigm*, *i.e.*, the models are trained on a labeled dataset, and in *the context of balanced anomaly detection*, *i.e.*, we detect anomalies on a time series data set where the normal and abnormal classes are almost equally represented.

Dataset description We have compiled a dataset consisting of 42 two-class time series datasets from the UCR archive (Dau et al., 2019) at the URL https://www.cs.ucr.edu/~eamonn/time_series_data/. The training datasets are balanced and have been preprocessed, with the notable absence of missing values and all data being z-normalized, following the

preprocessing described in (Dau et al., 2019).

Model description For the automation level 1, we have chosen to investigate the performance of four supervised classifiers: Random Forests, Naive Bayes, AdaBoost, and MLP, since they are also considered by the AutoML tool (of automation level 2). We have employed the scikit-learn implementations of these models (Pedregosa et al., 2011).

5.2.1.2 Scenario 2

In the second scenario, we evaluate automation level 1 and automation level 2 in *a supervised paradigm* and in *the context of unbalanced anomaly detection*, *i.e.*, we detect anomalies on a time series dataset where the normal data proportion is higher than the anomalous data.

Dataset description We have compiled 20 time series datasets selected from the 42 datasets from UCR repository (Dau et al., 2019), used in scenario 1. These datasets have been selected to have a training set with more than 100 observations. We then created an imbalanced version for all these datasets by decimating the anomaly class, the majority class is set to be 90% of the training set and the minority class set to be 10% in each training set. We have also added a dataset originally unbalanced in the UCR repository, with 90% majority class and 10% minority class, resulting in a total of 21 datasets.

Model description For the automation level 1, we have chosen to investigate the performance of the classifiers previously selected in scenario 1. We have also added a One Class SVM, trained on the majority class, and an outlier detection algorithm, Isolation Forests, trained on the entire training set. We have used the scikit-learn implementations for both One Class SVM and Isolation Forest.

5.2.1.3 Scenario 3

In the third scenario we evaluate automation level 1 and automation level 2 in a semi-supervised paradigm (*i.e.*, the models are trained on a dataset that contains both labeled and unlabeled data) and in the context of unbalanced anomaly detection.

Dataset description For this scenario, we have modified the 20 time series datasets from scenario 2 by preserving 30% of the labels in the training set and treating the remaining data as unlabelled.

Model description For the automation level 1 anomaly detection task, we have chosen to investigate the performance of the classifiers previously explored in Scenario 1 - automation level 1, but this time on the labeled data only. Furthermore, we have also trained the semi-supervised and outlier detection workflows from scenario 2 on the entire training set, including the unlabelled data.

5.2.1.4 Scenario 4

XPTOKN

In the fourth scenario, we evaluate also automation level 0 where the data scientists designs a machine learning workflow based on his/her understanding of the business requirements and of the data, automation level 1 and automation level 2 in the context of detecting and localizing anomalies (*i.e.*, in the precedent scenarios the goal was to detect if a time series as whole contained anomalies, but in this scenario the goal is to localize precisely the anomalous pattern inside the time series.) The training paradigm we evaluate is supervised in the context of unbalanced anomaly detection.

Dataset description In this scenario, we aim at recreating an actual use-case presented by a partner company Ezako in which they want to detect damage in bearings based on motor current signature



Figure 5.2: Scenario 4 automation level 1 workflow steps

data. As we can not disclose the datasets from Ezako, we have generated datasets to simulate similar situations with similar constraints. The domain expert explains that a current time series from a healthy bearing is cyclical with a fixed period. In contrast, a damaged bearing will lead to data portions with different frequencies depending on the type of damage. The training and test set available to build the workflow comes from two devices that have anomalies with similar distortions in the current frequency. However, the domain expert explanation implied that the anomalies could appear with different frequency distortions. Therefore we added a second test that will not be available at the workflow building time to emulate how our selected workflow would deal with new data at deployment.

We generated three datasets of sinusoidal time series with a constant frequency of 1 Hz and added anomalous patterns of varying lengths. The three datasets consist of 90-10%, 93-7%, and 97-3% ratios normal vs anomalous patterns (or contaminations). These anomalous patterns are made up of changes in sinusoidal frequencies. We have built one training set and two test sets for each dataset. Test 1 contains anomalies similar to the anomalous patterns in the training set, with a sinusoidal frequency of 0.1 Hz, while test 2 contains anomalies with a different sinusoidal frequency of 4 Hz.

Model description For automation level 1, we apply our model selection via hyper-parameter optimization to the following algorithms: MLP Classifier, Random Forests, Naive Bayes, AdaBoost, One class SVM and Isolation Forest models.

For automation level 0, where the task is detection and localization of the anomalies within time series, the data scientist suggested reusing one of the previously developed workflows to solve an anomaly detection and localization task in submarine acoustic data. The workflow was promising since the anomalies' nature of frequency distortion in this scenario is similar to use case detecting and localizing acoustic data anomalies. The workflow highlights the frequency differences between the expected signal and the anomalous patterns by computing short-time Fourier transforms over fixed-length windows of the signals. The transformed data are then fed into a CNN-based Autoencoder (Kieu, Yang, & Jensen, 2018) see figure 5.2. The model is trained on the spectrograms of the healthy windows from the training time series. Furthermore, we use the available training labels to fine-tune the reconstruction error threshold⁴.

Training and evaluation details For this scenario the training time for all 3 levels of automation was under 20 minutes, and since we are looking for anomalous patterns in a time series (detection and localization), we monitored the minority class F1-score.

5.2.2 Summary table

⁴The CNN-Autoencoder specializes in reconstructing normal data. Consequently, when it attempts to reconstruct abnormal data, the dissimilarity between the original abnormal data and the reconstructed version is more pronounced compared to the reconstruction of normal data. A threshold is set to differentiate the normal from the abnormal data

Contamination	automation level	Task	Dataset	models	Training paradigms
Scenario 1	Level 1	Anomaly detection	42 time series from UCR benchmark	Random Forests, Naive Bayes, AdaBoost, and MLP.	Supervised
	Level 2			AutoML tool TPOT	Supervised
Scenario 2	Level 1	Anomaly detection	21 time series two-classes unbalanced	Random Forests, Naive Bayes, AdaBoost, MLP, One Class SVM, Isolation Forest	Supervised, semi-supervised and unsupervised
	Level 2			AutoML tool TPOT	Supervised
scenario 3	Level 1	Anomaly detection	21 time series two-classes unbalanced with 30% of labeled data	Random Forests, Naive Bayes, AdaBoost, and MLP, One class SVM, Isolation forests	semi-supervised,
	Level 2			AutoML tool TPOT	Supervised, semi-supervised, unsupervised
scenario 4	Level 0			CNN-Autoencoder workflow	Semi-supervised
	Level 1	Anomaly detection and localization	3 Bearing motor signature time series datasets	MLP Classifier, Random Forests, Naive Bayes, AdaBoost, and Isolation Forest models	Supervised, semi-supervised and Unsupervised
	Level 2			AutoML tool TPOT	Supervised

Table 5.1: Scenarios summary

5.2.3 Results description

resultsXP

Scenario 1 The results of the experiments indicate that the performances of the automation level 1 and automation level 2 approaches are comparable on standard time series datasets (Scenario 1), with the median F1-score around 0.83 and the third quartile F1-score slightly over 0.9 for both automation levels. However, automation level 1 had slightly higher scores in the first quartile F1-score, 5 points higher than automation level 2, and the minimum, which was also higher than automation level 2 see Figure 5.3.

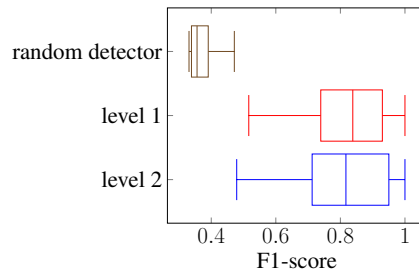


Figure 5.3: Scenario 1 results

fig_res_S1

The models pre-selected for automation level 1 were not chosen randomly; they were specifically chosen based on their suitability for the task. Additionally, the introduction of a time limit for both approaches meant that automation level 1 focused solely on optimizing well-suited models, while automation level 2 had to go through a much larger number of models, not all necessarily suited for the task, within the same time frame. As a result, automation level 1, with its narrower focus, yielded more consistent results, leading to a more compact box plot.

This support the idea that a focused subset of suitable models leads to higher overall performances.

Scenario 2 Here, the automation level 1 performed better than automation level 2 on several metrics. Automation level 1 scores were higher than automation level 2 on four metrics see figure 5.4, minimum (0.35, 0.47), first quartile (0.47, 0.57), median(0.70, 0.57), third quartile (0.88, 0.71), automation level 2 has a higher maximum score of 1 against 0.98. Both approaches perform better than their baseline, but there is a significant drop in performance compared to the first scenario on the same test sets.

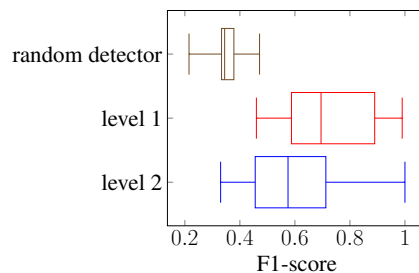


Figure 5.4: Scenario 2 results

fig_res_S2

The introduction of new semi-supervised and unsupervised models to the pool of models to go through in automation level 1 that are not available to the automation level 2 made the difference. The One Class SVM trained on normal data in automation level 1 outperformed supervised models from automation level 1 and the best results from automation level 2 under the time constraint we imposed.

Scenario 3 The automation level 1 scores better than automation level 2 in minimum (0.37, 0.43), median (0.68, 0.63), and first quartile (0.54, 0.5), while automation level 2 scores better than automation level 1 in the third quartile (0.83, 0.72) and maximum (1, 0.95) see figure 5.5. The performance of the automation level 1 approach deteriorates from Scenarios 1 to 2 and from 2 to 3, as expected. However, on the other hand, the automation level 2 approach surprisingly performs slightly better in the Scenario 3 compared to the Scenario 2.

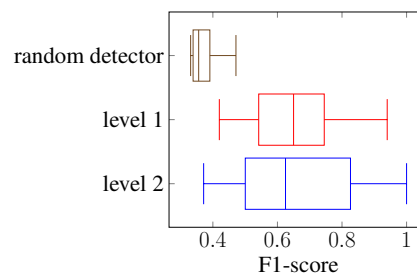


Figure 5.5: Scenario 3 results

fig_res_S3

The pool of supervised algorithms used in automation level 1 are benefiting only of the labeled data which represents a little part of all data available and a significantly lower amount of labeled data compared to Scenario 1. This explains why the supervised models perform poorly compared to the models generated by automation level 2. Additionally the training of One Class SVM model is conducted on whole dataset and proved to also produce poor results as it learns from highly unbalanced data.

Scenario 4 In the industrial use-case of anomaly detection in scenario 4, the automation level 0 approach incorporating domain knowledge performed much better than automation level 1 and automation level 2 on the test set with different anomaly frequencies. On test set 2, the automation level 1 and automation level 2 methods score 0 on the minority class for the three contamination levels, while level 0 retains high performances (0.93,0.92,0.9) see table 5.2.

In the three datasets, both automation level 1 and automation level 2 achieved high scores on the initial test set because they overly focused on the training data, causing overfitting. However, their performance declined on the second test set. This drop occurred because the anomalies maintained the same nature (such as changes in current frequency), but the specific values of the frequency distortion changed. This change couldn't be captured by simply analyzing the time series data.

Conversely, automation level 0 performed well across both test sets in all three datasets. This success was due to its ability to recognize spectrograms that highlighted the frequency patterns

of normal data behavior. Unlike automation levels 1 and 2, which learned the specific values of normal and abnormal sequences, automation level 0 focused on understanding the overall behavior represented in the spectrograms, leading to consistent performance across different datasets and test sets.

Contamination	Approach	Test 1	Test 2
97-3%	Level 0	0.89	0.9
	Level 1	0.96	0
	Level 2	0.97	0
93-7%	Level 0	0.93	0.92
	Level 1	0.97	0
	Level 2	0.97	0
90-10%	Level 0	0.98	0.93
	Level 1	0.96	0
	Level 2	0.95	0

Table 5.2: Scenario 4 results by approach and test set

Table 5.2 illustrates the importance of comprehending the limitations of utilizing an AutoML approach. It is crucial to acknowledge that the data may lack comprehensive information about itself. In the provided example, in test case 2 (the hidden test case during solution development), anomalies manifested in a distribution that the automatically generated model couldn't differentiate from normal behavior. This emphasizes the limitations of relying solely on automatic approaches guided by data.

5.2.4 Discussion

Overall, our results indicate that automation at levels 1 and 2 performed acceptably on standard time series datasets. In Scenarios 2 and 3, where the datasets were imbalanced, the difference in performance compared to the test random prediction baseline was relatively modest.

Furthermore, results from scenario 4 demonstrate that high performance on a test set can align differently from the specific requirements of domain experts. Therefore, only relying on automatic evaluations without a comprehensive understanding of the nuances of the use-case can result in the dangerous misconception that the workflow is functional when it is not.

Yet, automation came to solve some problems automation level 0 ("data-scientist-and-domain-expert") approach presents; the first one is to reduce the time that the "come and go" between the domain expert and the data scientist takes to understand the requirements and evaluate the results, and the second one is the unpredictability of the process: For instance, if the data scientist did not capture the whole nature of the anomalies, or if the data scientist did not stumble upon a similar use-case in previous experiences?

On the other hand, by leaving these tasks to automation when the use-case requirements are not "standard", the solutions we get may not be adequate, and worse, we do not even realize if and why such a use-case deviates from "standard".

Many of these issues on AI automation have been of interest to the software engineering community for years, encountered from the perspective of software product lines (SPLs) (Niehaus,

Pohl, & Böckle, 2005). SPLs introduce a set of notions to solve similar problems as in AI automation while retaining control over the solution generation process, facilitating the inclusion of domain experts in the loop, and helping manage the requirements and solutions which evolve through time. Even though the vocabulary might change, here are some notable correspondences between AI automation and SPLs:

Creating a machine learning workflow is similar to crafting a scientific workflow using SPL. Similarly, the process of composing and optimizing machine learning workflows is akin to the generation of products, as we will explore further in the following chapter.

In automated machine learning identifying the properties data wise that the meta learning model will use to reduce the space of possible machine learning artefacts to compose a workflow is assimilated in an SPL approach, to a domain engineering step that captures the variabilities and commonalities of the use case business requirements and data properties wise (Siegmond et al., 2012). In this aspect SPL's can consider the business requirements while meta learning can only consider the properties of the data.

Meta-learning extracts the rules data wise that guide workflow composition and optimization. In SPLs the equivalent of these rules are captured by an application engineering step and are explicitly formulated in the form of a constraint system that orchestrates the composition of a scientific workflow as shown in the example form section 4.3.

When using a classical machine learning process such as the ones presented in section 2.6.1 identifying similarities with past use-cases especially business requirements wise is done empirically by the data scientist, while SPL allows to match them with a configuration step accordingly to the identified criteria in the domain engineering phase (Niehaus et al., 2005; Amraoui et al., 2022; Nieke, Seidl, & Schuster, 2016).

5.2.5 Explicated knowledge

What we capture from this experiment underscores the importance of preemptively understanding the potential variance in data distribution before embarking on the design of an ML workflow, particularly when deciding between supervised and semi-supervised methods. Furthermore, in this specific context, the transformation of data into spectrograms using fast Fourier transform proved pivotal in distinguishing anomalies from regular observations, given their temporal occurrence within specific windows as opposed to isolated outlier points.

However, this experimentation incurred significant human time costs, as it entailed numerous trials and errors in the automation level 0 approach. To mitigate such time-consuming endeavors in future instances of similar use-cases, it is imperative to inquire about criteria that were previously overlooked before consulting domain experts. This proactive approach can facilitate the direct reuse of solutions, thereby minimizing human time loss.

The knowledge extracted from this experience is twofold: firstly, it underscores the necessity of determining whether the data distribution is expected to remain consistent in production before initiating the trial-and-error phase. Secondly, it highlights the efficacy of semi-supervised approaches in addressing issues arising from shifts in data distribution, as supervised methods often overfit on the accessible dataset during the design phase while semi-supervised methods do not.

5.3 Conclusion

In the process of generating machine learning workflows to solve a given problem, data scientists enquire about business requirements analyze the data and then come up with a solution through trial and error. To decide if a trial is a success or a failure, they assess it with ML related metrics such as accuracy, recall, f-score, ROC curve and so on. If these metrics render what they deem a good score on the available test set at design, it is a success, otherwise it is a failure or an error. However they do not necessarily formulate why a given workflow failed before testing a new one. And the number of trials can be high because the search is not guided and only influenced by the preferences of the data scientist conducting the search and his/her given knowledge. The approach is intuitive and therefore its entropy is high *i.e.*, for data scientist A it takes 3 trials to find an adequate solution while for data scientist B it takes up to 15 trials. Moreover they evaluate their solutions on a known test set and cannot know from the data alone if the actual data in production is similar in properties to the test set with regards to the deployed solution as seen in Scenario 4, which creates an additional monitoring phase when the model is deployed, and in the case where the workflow performs poorly, the whole process has to be restarted taking into account the new data where it failed. Additionally, each data scientist learns from their own experiences only, *i.e.*, data scientist A produces a wrong workflow, understands why it failed and takes this into account the next time they deal with similar use cases, and data scientist B does the same thing, each in his/her own bubble, without sharing the knowledge acquired. Data scientist B could go through the same mistakes data scientist A went through and vice-versa. Moreover even the notion of similarity of use-cases we use, is subjective to a given data scientist because they might make conciliation that others might oversee even when inquiring about the problem itself from the domain expert since they are the ones translating business problem into business requirements. The global process is not structured enough and remains heavily reliant on who/what steers it (*i.e.*, the data scientists intrinsic knowledge).

In chapter 6, we propose our approach to diminish the entropy of the process by mutualizing the acquired knowledge into a common knowledge base that is built from the trial and errors of each data scientist to help them steer the process of machine learning workflow generation more efficiently and utilize each others knowledge. In Chapter 7, we look at how to exploit the information contained in the knowledge base, which adds to existing knowledge and has an impact on the generation of workflows for new problems, as well as on the correction of previous workflows.

Contribution 1. *Contrary to current ML practice, we have shown experimentally that automation is not the answer in all contexts. Not all information is in the data.*

CHAPTER 6

Contextualizing ML variability modeling

chap_genWF

6.1	Introduction	63
6.2	Approach overview	64
6.2.1	Contextualized Solutions within the Overall Solution Space	64
6.2.2	Process overvie	66
6.2.3	Tooling overvie	67
6.3	Contextualized ML Solution Variability Modeling	68
6.3.1	Multi-Domain Variability Modeling: Supporting Contextualized Solution Space Reduction	68
6.3.2	Modeling Applications and Code Interactions for Efficient Solution Retrieval	70
6.3.3	Formal definition of the interaction	71
6.4	Applications	72
6.4.1	Scenario 1: Retrieve and clone notebook	72
6.4.2	Scenario 2: Generate a new notebook	73
6.4.3	Industrial case stud	73
6.4.3.1	Domain variability model	76
6.4.3.2	Variability model of realized products and assets	76
6.4.3.3	First results on the usefulness of configurations in determining workflow suitability	77
6.5	Conclusion	78

6.1 Introduction

Building effective data science systems is a challenging sociotechnical endeavor involving technical and human work (Passi & Sengers, 2020). Designing a universal solution that can work for any Machine Learning (ML) problem specification is not possible due to the high combinatorial diversity of both data (Bilalli, Abelló, & Aluja-Banet, 2017) and business requirements (Habibullah, Gay, & Horkoff, 2023). Additionally, the pool of available ML solutions is continuously growing, making it increasingly difficult for data scientists to navigate this "wilderness" of possibilities (Fernández-Delgado, Cernadas, Barro, & Amorim, 2014; Zaharia et al., 2018). Consequently, a data scientist may, for the same problem, try several preprocessing libraries, several types of models (e.g., decision trees and neural networks), and even multiple frameworks for the same type of model (e.g., TensorFlow and PyTorch) (Zaharia et al., 2018).

⇒ *This chapter explores assisting data scientists in characterizing their requirements and algorithmic preferences (**Takeaway 1, Takeaway 4, Takeaway 13**) (cf. section 6.2). Our approach involves guiding them through a step-by-step reduction of the solution space (**Takeaway 6, Takeaway 8, Takeaway 9, Takeaway 10**) (cf. section 6.3.1).*

A recent study (Koenzen et al., 2020) highlights the importance of code reuse in ML solutions production, with 18% of participants' time spent searching for code samples online in platforms . Duplication and reuse of code rely on tutorials or APIs and various sources such as GitHub or GitLab, because solutions are problem-sensitive (*i.e.*, sensitive to initial data properties and domain expert business requirements) , and the field is evolving rapidly. Thus, widely used platforms such as OpenML (Vanschoren, van Rijn, Bischl, & Torgo, 2013) or ModelDB (McDougal et al., 2017) allow the specific sharing and discovery of ML workflows by scientists. Notebook sharing, for instance, is widely supported by popular platforms such as Kaggle¹ (over 400K notebooks today), Google Colab², Jupyter Notebook, JupyterLab³, or Baidu AI Studio⁴. To the best of our knowledge, search options to retrieve ML workflow code are mostly based on a textual search (on domain keywords, benchmarks/competitions, machine learning artifacts, language, author, or dataset), possibly using ontologies as in the case of ModelDB (McDougal et al., 2017). Such search options could lead to various types of errors. For example, in the case of searching notebooks by keywords, this can result in an overwhelming number of notebooks, some of which are not even related to the field, without the problem (*i.e.*, initial data properties and business requirements) being addressed.

⇒ *In this chapter, we address the issue of searching for past solutions (limited to notebooks), incorporating not only considerations of properties related to data and algorithms but also the context relative to **Takeaway 11** and **Takeaway 12** (cf. section 6.3.3 and section 6.3.2).*

Additionally when framing an ML problem data and business requirements wise in most cases the framing is only partial as some insight remain unknown to the data scientist.

⇒ *In this chapter we delve into handling partial knowledge of the ML problem through an extension of configurations management and incremental understanding of the ML problem through iterative trial and errors and continous feedback relatively to **Takeaway 5** (cf. section 6.3.3). The data scientists also require assistance in retrieving code artefacts to speed up their trial and error*

¹<https://kaggle.com/>

²<https://colab.research.google.com/>

³<https://jupyter.org/>

⁴<https://aistudio.baidu.com/aistudio/index>

process as we facilitate the sharing of past code artefacts, ml problem and solutions relatively to and **Takeaway 2** (cf. section 6.3.2)

We start this chapter by presenting our approach overview (cf. section 6.2), then we position our work in regard to contextualized machine learning solution (cf. section 6.3), then we present an illustration of our approach (cf. section 6.4) and conclude.

6.2 Approach overview

We begin by contextualizing our work within the context of solution space reduction (cf. section 6.2.1). Next, we introduce the proposed process aimed at assisting data scientists in shaping an ML workflow (cf. section 6.2.2). Finally, we provide an overview of the tool we developed (cf. section 6.2.3).

6.2.1 Contextualized Solutions within the Overall Solution Space

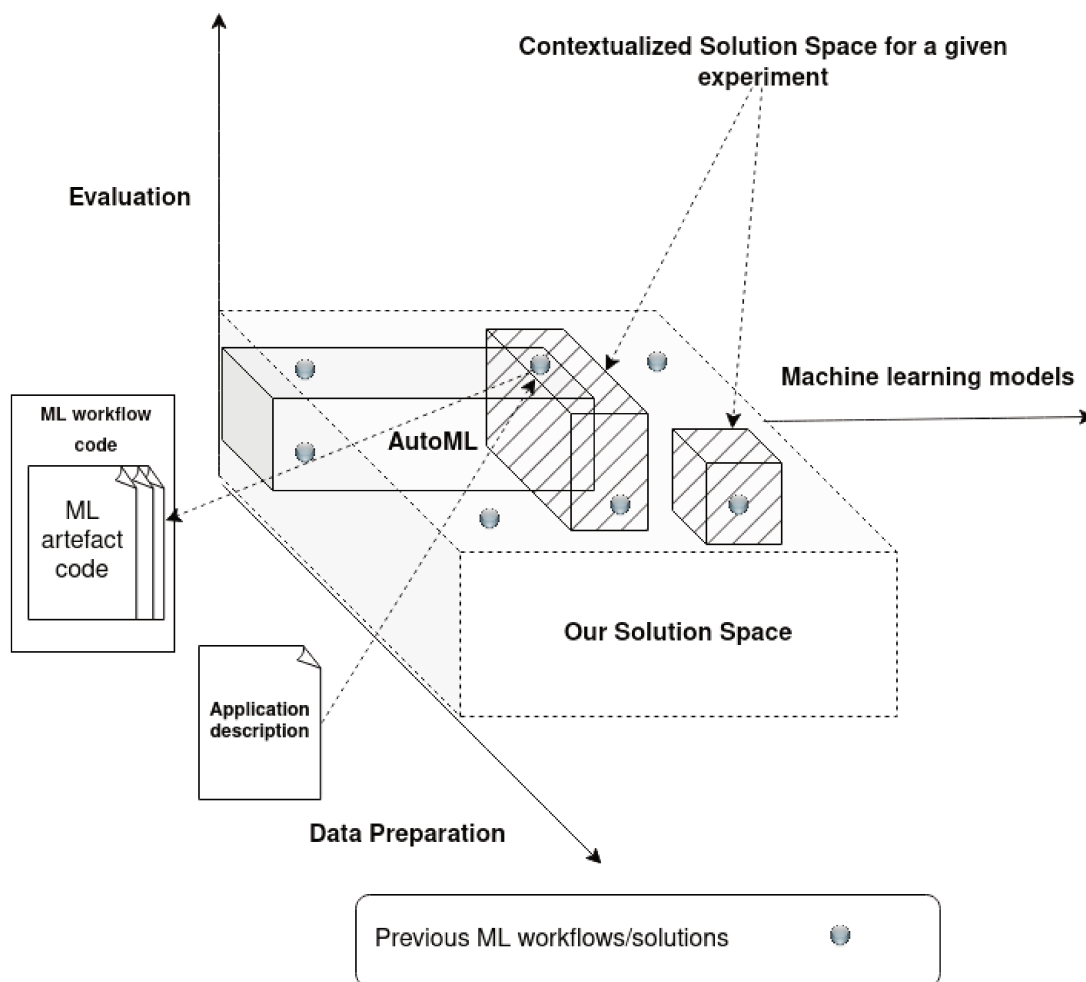


Figure 6.1: Contextualized Solutions within the Overall Solution Space

Figure 6.1 provides a visual representation of our approach to tackling the challenge of navigating a solution space within the context of an experiment. An experiment in this context encompasses a machine learning problem business requirements-wise, its associated data, and potential solutions. Here are some definitions pertaining to the understanding of the figures.

`Our solution space` represents the entirety of machine learning workflow components, encompassing both the previous ML workflows stored in our knowledge base and new possibilities. Initially unconstrained, it constitutes the starting point for solution exploration.

`Previous ML workflow solutions` refers to the previous machine learning workflows that have been produced and retained within our knowledge base that we call ML workflows product) (see section 6.3.2 for reference to ML workflows products).

`ML artifacts code` corresponds to the tangible implementations of ML artefacts components (see section 6.3.2 for reference to ML artefacts).

`Application description` relates to the detailed representation of a specific problem and its corresponding solution within a given application. This depiction encompasses criteria including business requirements, data attributes, and the constituent machine learning artifacts of the solution machine learning workflow we refer to it as Applications products in section 6.3.2.

`AutoML` depicted in the figure denotes the automated machine learning solutions contained in our knowledge base.

The patterned cuboid within the diagram signifies a more constrained solution region, termed the "contextualized solution space for experiments." This cuboid encapsulates the set of feasible solutions that align with the specific requirements of the experiment.

Additionally, it's worth noting that we've chosen to portray the solution space across three fundamental dimensions: variations in machine learning models, methods for evaluating success, and strategies for data preparation. Nevertheless, it's worth considering that further dimensions exist that could significantly enhance the precision of this visual depiction.

A notable aspect pertains to our use of the term "AutoML." As we want to clarify, our reference to "AutoML" is confined to specific AutoML tools that we've meticulously evaluated within the bounds of our solution space. This demarcation is crucial to avoid any misconceptions, as the broader notion of the general AutoML landscape extends beyond the scope of our focused considerations. Therefore, we've positioned our interpretation of AutoML within the contours of our solution space.

Our Solution Space evolves over time by adding new knowledge, such as new algorithms or new composition rules and evaluation methods and other dimensions not depicted in the figure that might be of importance, such as *MLOPS* considerations that we did not include in the versions presented in this work.

The *hatched boxes* then represent solution spaces that are tailored for specific applications and are encompassed within our overall solution space. We refer to these as contextualized solutions because they also consider the business requirements. As discussed in section 2.6.2 of the state of the art, AutoML tools serve as effective means to reduce the solution space based on problem data. Hence, AutoML solutions for a given problem can be encompassed within our proposed solution space. Conversely, there might be situations where no AutoML solution is well-suited for specific applications.

Our contributions to this vision, focus on: **1)** providing support to the data scientist in analyzing their problem and generating an adapted ML workflow, **2)** modeling the interactions between the solution space and the addressed problems, considering business requirements, and **3)** identifying similarities with previous solutions and applications when encountering new problems.

6.2.2 Process overview

To empower domain experts in designing problem-tailored solutions, we propose a process consisting of the following steps, as illustrated in fig. 6.2.

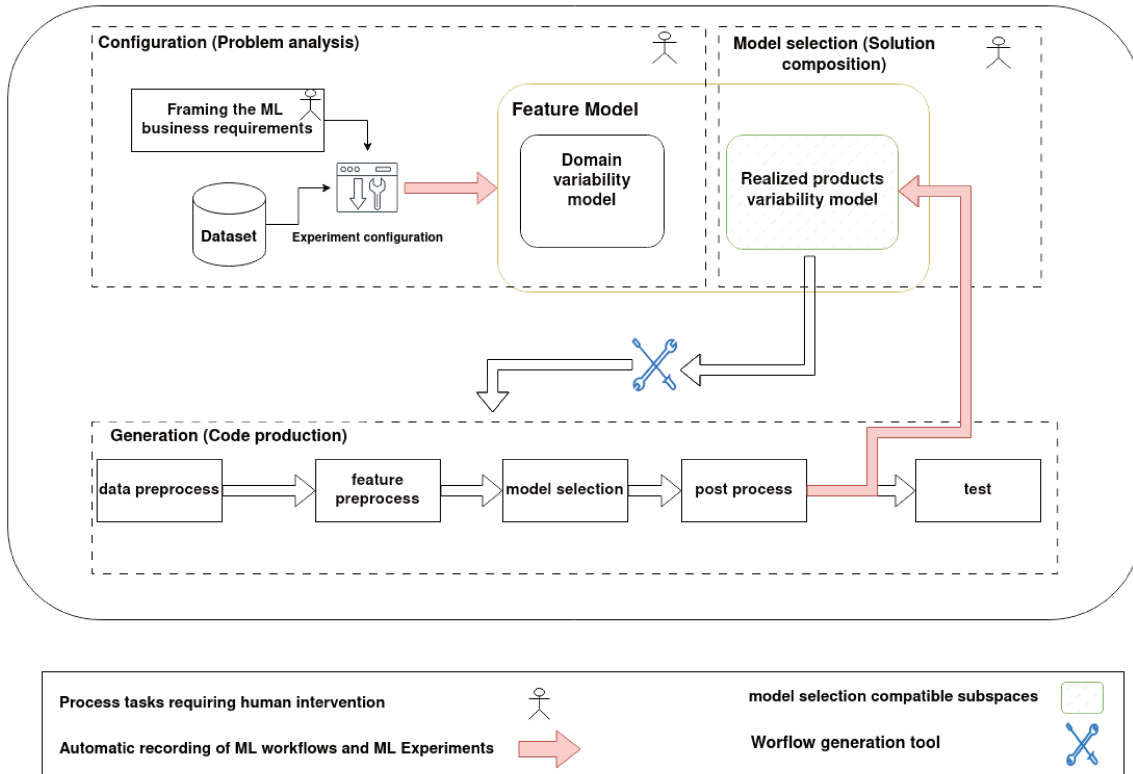


Figure 6.2: Process overview

Problem analysis: Configuration Step During the problem analysis phase, the data scientist relies on the "configuration" of the requirements to gather relevant information for analysis. This step includes extracting metadata from the data and discussing key questions with the domain expert to better define the ideal solution. The system actively guides the data scientist, utilizing her responses and dataset metadata to narrow the questions progressively. This flexibility enables her to answer in the order that suits her, for instance, choosing algorithm families first and then detailing the necessity of embedding the generated model. According to the provided answers, the system efficiently eliminates unsuitable artifacts, enabling the composition of a customized machine-learning workflow for the optimal solution. However, achieving comprehensive metadata about the data and a thorough analysis of business requirements poses challenges. Analyzing metadata can be resource-intensive; for instance, OpenML defines more than a hundred "meta-features" (Bilalli et al., 2017), whose valuations are only helpful based on the problem to solve. Simultaneously, requirements analysis depends on expert domain knowledge, the datasets' characteristics, and the algorithms' availability. Recognizing the intricacies of defining business requirements by domain experts, where the response to a given question may be 'I don't know,' is crucial. As a result, our approach acknowledges the inherently incomplete nature of our knowledge

in analyzing the problem at hand. It incorporates reasoning that considers partial configurations, and this early recognition forms a pivotal element of our contributions.

Solution Composition: Model selection Step The solution composition step entails selecting machine learning artifacts that are relevant to the problem in question. This process can take place simultaneously with the problem analysis. It is common for data scientists to autonomously choose specific algorithms. Consistency checking enables them to ensure that their choices align with the pursuit of an effective solution.

During this step, data scientists often seek to leverage solutions that have been previously used for similar problems. Based on the configuration of the desired solution, the system suggests past applications corresponding to the current configuration or solutions that use compatible machine learning artifacts with the description of the problem from the configuration step.

Code Production: Generation Step From the selected artifacts, the codes corresponding to the solution are generated. These codes incorporate calls to the selected artifact codes. The generation process relies on a model engineering approach (Schmidt, 2006). When it comes to reusing existing codes selected by the data scientist, it currently involves a simple copy of the previous codes.

Data scientists can then store their problem configuration, allowing them to revisit it later. They then appropriate the generated code by adapting it according to the results.

The functional overview of the process is depicted in the fig. 6.3:

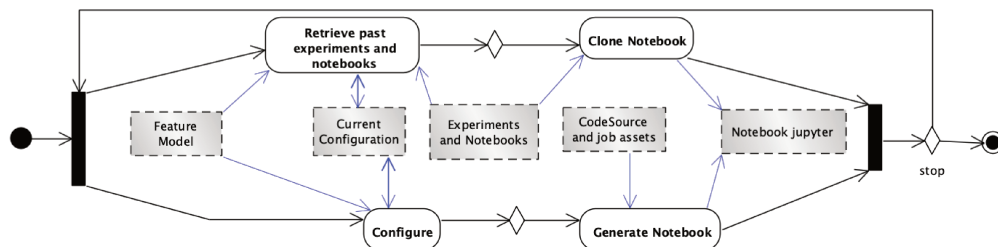


Figure 6.3: Functional overview of our approach

:processmodeling

ss_tool

6.2.3 Tooling overvie

Définition 6.2.1.

In fig. 6.4, the data scientist chooses the feature related to a problem in the business requirements window. This problem involves the expected behavior of the data potentially shifting in production. By doing so, the data scientist can prevent the recurrence of errors previously discussed in section 5.2.4

Figure 6.4 depicts the tool interface developed based on Elias Kuitert's feature-configurator⁵. Data scientists use the *Initial Data*, *Business Requirements*, and *ML Artifacts* tabs to configure their experiments. Visualization, including questions, is achieved by associating features with questions through a CSV file. The tool provides access to past experiments and notebooks through

⁵<https://github.com/ekuitert/feature-configurator>

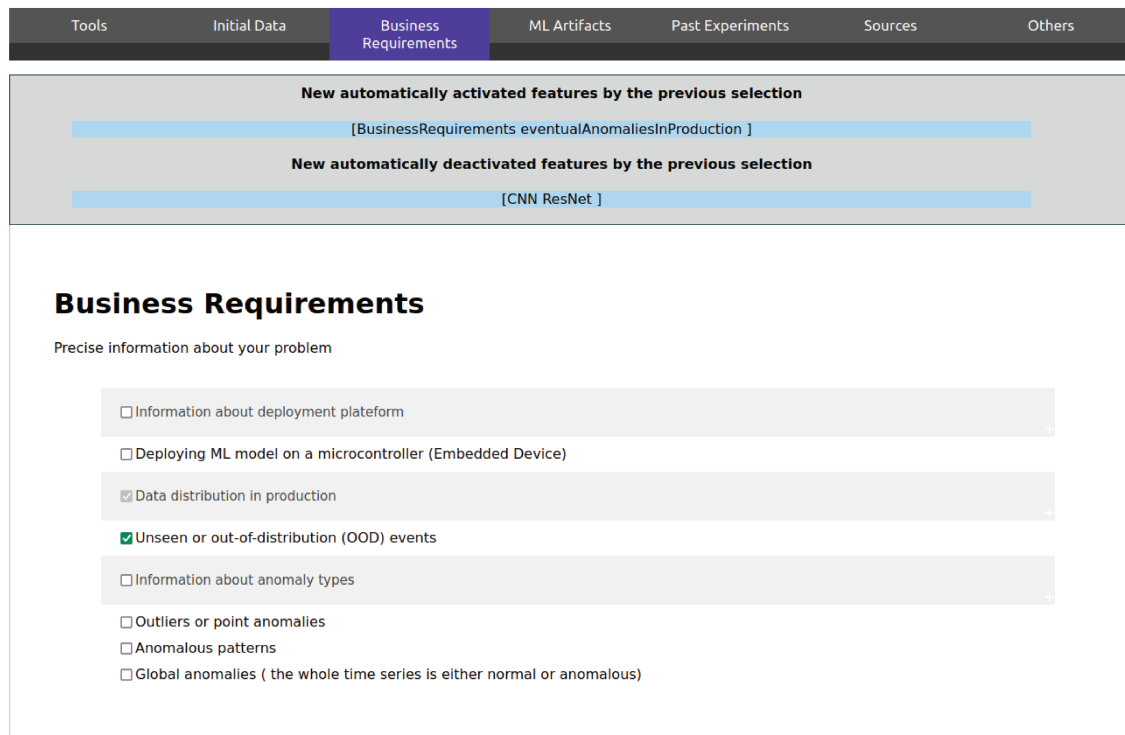


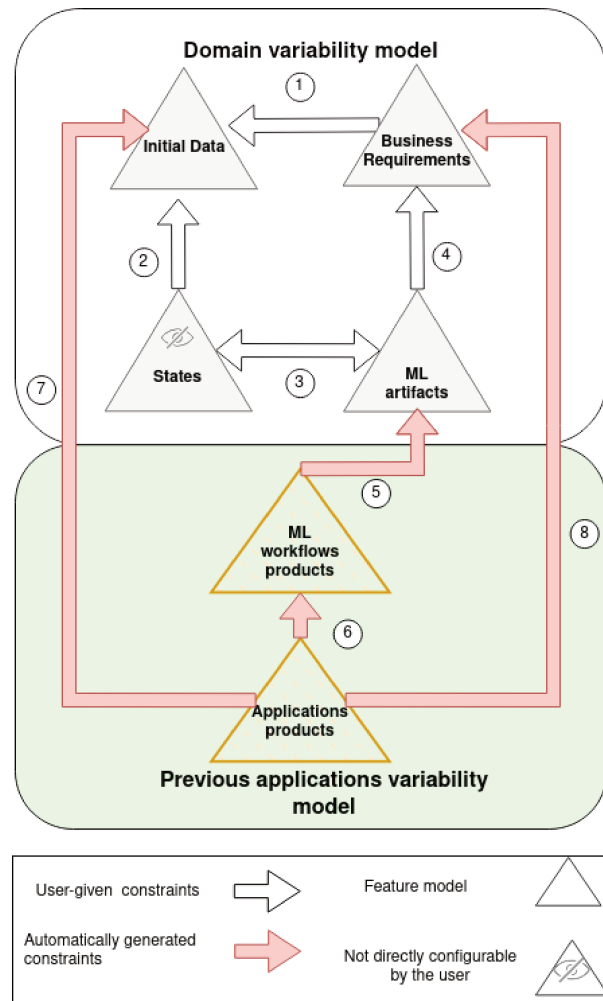
Figure 6.4: Tool UI for Business Requirements configuration

the *Past Experiments* tab. The generation step is accessible through the *Tools* Tab. Technically, generating a totally new notebook is handled as such: Once the user requests from the tool to generate the notebook, the *generator* is only given the `MLArtifacts` part of the configuration. It then uses the mapping of the concerned submodel to `Job Assets` to find the requested product implementations. Those implementations are then copied into a data structure representing the notebook cells. This data structure is provided by Jupyter Python library designed to programmatically generate Jupyter Notebook. It must be noted that the choice of algorithms and ML workflows is considered part of the configuration, allowing data scientists to select established algorithms (Jordan & Mitchell, 2015). This configuration at a level usually considered as low (Dhungana et al., 2011) then can automatically drive the configuration of business requirements and vice versa.

6.3 Contextualized ML Solution Variability Modeling

6.3.1 Multi-Domain Variability Modeling: Supporting Contextualized Solution Space Reduction

This section presents the main principles that guide the representation of knowledge required for the configuration and selection steps. Some challenges related to AI automation have long been of interest to the software engineering community, aiming to generate complex code from desired solution configurations. Software Product Line (SPL) introduces a set of concepts to address similar problems to AI automation while maintaining control over the solution generation process,



fig_FMspaces

Figure 6.5: Knowledge partitioning into feature model spaces

facilitating the involvement of domain experts in the loop, and assisting in managing requirements and solutions both evolving over time (Niehaus et al., 2005). Thus, the role of meta-learning in extracting rules and guiding the optimization of the pipeline is captured by a domain engineering phase (Siegmond et al., 2012). This phase produces a constraint-based system to orchestrate the interoperability of ML artifacts. Building on techniques for representing the variability in this domain, we represented the domain variability using feature models (FM) (Benavides, Segura, & Ruiz-Cortés, 2010).

By categorizing the elements related to the different stages of ML workflow generation into six distinct spaces — initial data properties, business requirements, ML algorithms that we denote MLArtifacts composing a workflow, data transition states, and previously generated ML workflows that we denote ML workflows products and a comprehensive representation of the entire experience that we denote Applications products — we establish a structured framework as depicted in fig. 6.5. This framework defines the rules governing the interactions among these spaces during the selection process. In the subsequent sections, we will delve into the internal character-

istics of each space and elucidate the rules that regulate their interactions, both within and across spaces.

Initial Data Feature Model This Feature Model (FM) represents the space of initial datasets, specifically for time series data in our industrial partner’s domain. Certain properties, such as sampling frequency, dimensionality, and stationarity, can be automatically extracted from the dataset. However, other properties, such as handling missing values, require interacting with the business expert.

Consequently, the data scientist configures this sub-model by combining automatic data analysis and manually selecting relevant properties.

Business Requirement Feature Model This FM captures various requirements related to memory usage, model embedding, and providing explanations. The data scientist manually configures this sub-model by selecting the relevant requirements.

While the initial data space does not restrict other feature models, specificities regarding the initial data, such as sampling rates, can be included in the business requirements. This dependency is illustrated by *arrow 1* in fig. 6.5.

ML Feature Model This FM captures the ML artifacts comprising an ML workflow, such as algorithms, preprocessing techniques, evaluation methods, and workflow types. It is configured by the data scientist based on their specific problem and preferences.

States Feature Model This FM represents the different stages through which data progresses, with constraints reflecting the preconditions and impact of ML artifacts on each state. For example, an algorithm may require the data to be in a scaled state without necessarily requiring that state to be the initial one. The configurations of *Initial Data FM* and *ML FM* drive the configuration of this FM. Introducing the intermediate *States FM* allows us to handle knowledge evolution and to incorporate new algorithms without disrupting the constraints between states and ML artifacts. This separation is crucial as new data preparation algorithms are discovered and as the strengths and weaknesses of existing ones are revealed.

ML artifacts must satisfy preconditions and postconditions relevant to business requirements and data characteristics. The goal is to narrow the solution space to ML artifacts that fulfill these requirements, enabling the evolution of business requirements and solutions. The separation of concerns in the *ML FM* and *Business Requirements FM* (shown in *arrow 4* in fig. 6.5) facilitates this management of evolution. Additionally, the state of the data depends not only on its initial state (*arrow 2*) but also on its transformations and processing throughout the ML pipeline (*arrow 3*). The intermediate *States FM* plays a critical role in managing this variability and ensuring the overall effectiveness of the ML solution.

6.3.2 Modeling Applications and Code Interactions for Efficient Solution Retrieval

We leverage the terminology introduced by Idowu et al. (Idowu, Struber, & Berger, 2021) to present the assets that enable the retrieval of past applications and their associated ML workflow code or generate new code for a new application. We conceptualize past applications and their associated ML workflow code as products within the Software Product Line framework, specifically

the **Applications products** and **ML Workflows products** (see fig. 6.5). This structuring divides the application space into two variability subspaces, facilitating the identification of past applications and the potential cloning of their associated code. Other variability characteristics, such as datasets, application’s authors, and source scientific articles, are not depicted in fig. 6.5, although they help select past applications. However, they are not decisive in establishing a correspondence between the specifications of an expected solution and previous applications, which is the focus of this chapter.

Source code and job assets are used to generate code corresponding to mappings between the features of the *ML Feature Model* and these artifacts.

The constraints depicted by *arrows 5, 6, 7, 8* support identifying *applications* and *ML workflows* based on Domain variability model configuration. Our objective is to simplify the identification and selection of past solutions that conform to the current configuration. However, the absence of past applications should not restrict the exploration of alternative solutions. Hence, the interactions between workflows and ML artifacts should only narrow the search space for applications based on the current specifications. The non-use of an algorithm in a workflow should not eliminate it, as it can still serve as a valuable reference. Thus, we chose to represent only the relationships with the ML artifacts used by workflows, excluding their non-use and eliminating workflows that employ unsuitable algorithms (*arrow 5*).

An ML Workflow is associated with an application. One challenge is to accurately identify the context of a workflow, not only based on keywords but through a combination of factors, including data, business requirements, and the solution. It is important to identify situations where a workflow still belongs to the space of possible solutions, even if the application was not tackled in the same context. Therefore, we consider the workflow as an integral part of the application (*arrow 6*), while *arrows 7 and 8* depict the interactions with the initial data space and business requirements, respectively.

All constraints that drive this interplay between the four FMs capturing domain variability are provided by data scientists (indicated by white arrows in fig. 6.5). All other constraints from the two FMs capturing previous applications’ variability are automatically generated (indicated by red arrows in fig. 6.5).

In the next section, we exhibit our approach for modeling the interactions between applications and code, intending to improve the efficiency of solution finding. Using the software product line framework, we treated past applications and their associated ML workflow code as products within the SPL. This structuring, combined with the representation of ML artifacts and workflow relationships, allows for the identification of relevant past applications and the exclusion of workflows that use unsuitable algorithms.

ss_formel

6.3.3 Formal definition of the interaction

We provide here a formal explanation of how interactions with the realized product variability model are defined.

Domain variability model Configurations are our primary tool for determining *product application* context. Let $\llbracket FM_D \rrbracket$ be the set of valid configurations of the domain variability model FM_D .

In line with the domain modeling, a configuration c is composed of four subsets:

$$\forall c \in \llbracket FM \rrbracket, c = \text{initData}(c) \cup \text{busReq}(c) \cup \text{mlArt}(c) \cup \text{states}(c).$$

Partial configurations to specify ML problems As we work on *product applications* whose context is partially known and the SPL evolves, some features about the initial data or business requirements can be neither selected nor deselected, they are simply "unknown."

A set of features s in a configuration can then be defined as three subsets $s = selected(s) \cup deselected(s) \cup undefined(s)$. The intersection is empty between these three subsets.

ormalization

Experiments An *Product application* a is defined by a name n_a , the reference to a notebook nb_a and a valid configuration c_a , we note $a = (n_a, nb_a, c_a)$ where $c_a \in \llbracket FMD \rrbracket$

The subsets $initData(c_a)$ and $busReq(c_a)$ may contain undefined features. In contrast, $mlArt(c) \cup states(c)$ is complete; all the features are selected or deselected.

Variability model of realized products The domain of realized products is defined by a set of product applications, A . The constraint system uses the product application's name and the notebook's name to select or deselect them, depending on the other features that characterize the problem to be solved. The defined interactions are as follows.

$\forall a_i \in A, a_i = (n_{a_i}, nb_{a_i}, c_{a_i}),$
 (arrow 6) $n_{a_i} \Rightarrow nb_{a_i},$
 (arrow 5) $nb_{a_i} \Rightarrow \Lambda selected(mlArt(c_{a_i})),$
 (arrow 7) $n_{a_i} \Rightarrow \Lambda selected(initData(c_{a_i})) \cup deselected(initData(c_{a_i})),$
 (arrow 8) $n_{a_i} \Rightarrow \Lambda selected(busReq(c_{a_i})) \cup deselected(busReq(c_{a_i})).$

The constraint corresponding to arrow 5 deselects a notebook when an artifact used by the notebook can no longer be selected and, therefore, the product application associated with it (arrow 6). By restricting this constraint to the selected elements, it is possible to keep a notebook as an aid to clone even though it lacks components. The constraint (arrow 6) separates the deselection of the product application relative to the notebook from the deselection of the notebook. The two last constraints establish the correspondence between the experiment, the specification of the business requirements, and the initial data. Adding product applications builds all the above constraints automatically.

6.4 Applications

_evaluation

6.4.1 Scenario 1: Retrieve and clone notebook

s_scenario1

Objectives With this scenario, we want to illustrate how our approach differentiates itself from current practices. To determine whether a solution to a similar problem already exists, the user relies on her expert's insight. Once the constraints are applied, suitable solutions are proposed, enabling the user to choose an experiment or ML components.

Unfolding the scenario: Lea, a data scientist, is tasked with solving an anomaly detection problem on a dataset of motor vibrations. She must produce a solution that raises alarms when abnormal patterns indicate a defect in the motor bearing:

Lea first configures the `InitialData` FM (cf. Figure 6.7a), specifying the data type is `TimeSeries`, `PartiallyLabelled`, and not `NormalizedData`. *The given specification is sufficient to exclude Experiment XP1, which dealt with motor vibration but was performed on*

unlabeled data. However, this finding does not contradict the associated notebook, which employs an algorithm capable of handling partially labeled data. The constraints represented by arrow 8 in Figure 6.5 were applied. Lea selects in the BusinessRequirements FM (cf. Figure 6.7b) `NovelAnomaliesEmergeInProd` and identifies the anomalies to be detected as `patternAnomaly`. Based on the current state of the configuration, CNN and Resnet algorithms cannot be selected anymore, as they are not suitable for handling new anomalies in production (according to arrow 4 in Figure 6.5, i.e., constraints 6 and 7 in Figure 6.7e). At this stage, several experiments are compatible with the current configuration. Lea then clones the experiment XP2 to work with it. As a result, she has been able to retrieve several past experiments and notebooks by giving the system her configuration. Thanks to the constraints model she has been able to clone one that matches her problem.

ss_scenario3

6.4.2 Scenario 2: Generate a new notebook

Objectives With this scenario, we want to illustrate the correctness of the generative process, from the proposed code artifacts to the generated notebook. In the case where none of the experiments nor notebooks are reusable, we want to validate that our approach leverages constraint system propagation to reduce the number of unsuitable component compositions.

Unfolding the scenario Lea is tasked with solving another anomaly detection problem. This time she has to use a collection of inline process control measurements from various sensors during the processing of silicon wafers for semiconductor fabrication. She must build a solution that raises the alarm for faulty wafers.

Lea informs the data properties as being `TimeSeries` and `FullyLabelled`. She also deselects `NormalizedData` as the data are not normalized. *No experiments are compatible with her data configuration.* As Lea knows that no new anomalies will arise during the production use, she then deselects `NovelAnomaliesEmergeInProd` in the business requirements. Following a discussion with the expert, she is able to determine that the series is irregular. Thus she checks `globalAnomaly`. *The two still available notebooks do not fully meet the new requirements. The tool suggests available algorithms that Lea can select.* To compare the two models, Lea generates two notebooks that share a `MinMax` scaling preprocessing, one notebook featuring a CNN classifier and the other featuring a Resnet classifier. *She chose to conduct a generative process by picking machine learning components. Only configuration-suitable features were provided to her. In order to test several classifiers, Lea had to generate as many notebooks.*

ss_industrialUC

6.4.3 Industrial case stud

Our collaborative partner is a well-established data science SME with extensive experience identifying anomalies within clients' time series. They rely on computational notebooks to perform exploratory tasks such as data visualization, processing, and workflow building/testing. Using their anomaly detection software, they build machine learning workflows for deployment in production, which they refer to as **Legacy Workflow Representations Products** (LWRP). In this section, we report on the application of our approach to switch the internal process of the company, from a keyword-based search inside the set of past experiments (LWRP) to a configuration process.

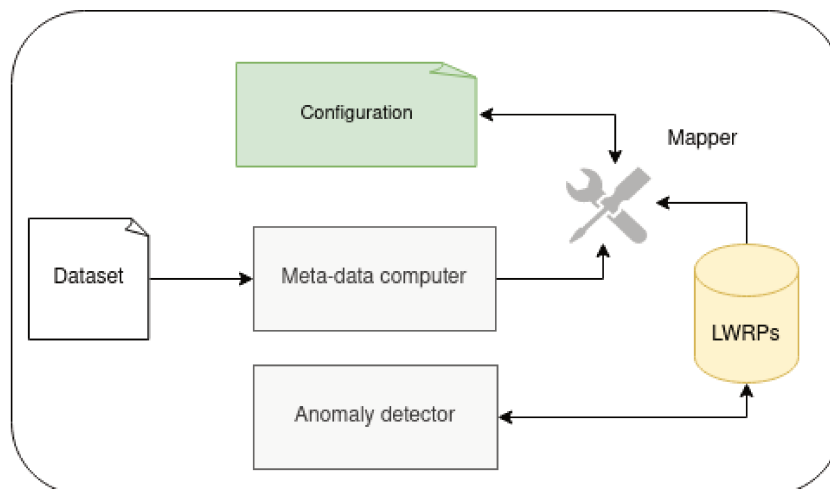


Figure 6.6: Company tool overview

fig_upalgo

Company software components In Figure 6.6 we depict the main components of the company’s software.

The `Meta-data computer` module is responsible for computing various properties of datasets. It automatically detects some properties, such as the existence of missing values, while others that are computationally expensive⁶ are only calculated on demand, based on the data scientist’s request. For instance, the module can check for time series stationarity, but only when explicitly instructed to do so by the data-scientist.

The `Mapper` module creates mappings between the various components of the software. Specifically, it performs two main tasks:

- (i) It takes the output of the `Meta-data computer` module and transcribes it into the selection or deselection of initial data features in the configuration.
- (ii) It creates an `LWRP` from a configuration and vice-versa.

The `anomaly detector` module has several functionalities:

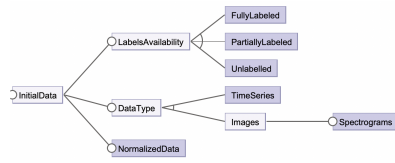
- (i) It generates machine learning workflows from an `LWRP`.
- (ii) It enables editing an `LWRP`.
- (iii) It generates a new `LWRP` from the selected ML artifacts.

To evaluate our approach, we discuss two illustrative scenarios, each one corresponding to a real-world case that has been simplified to clarify the propagation principles. Still, they enable us to demonstrate the capabilities of our approach.

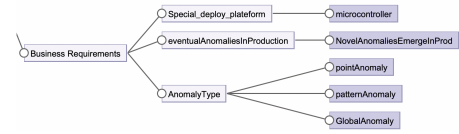
As illustrated in Figure 6.7c, the solution space comprises three primary types of solution components (preprocessing elements, algorithm components, and postprocessing components).

As depicted in Figure 6.7d, the state space structure is similar to the one of the initial data space. However, it captures the current state of the data, which differs from the former in terms

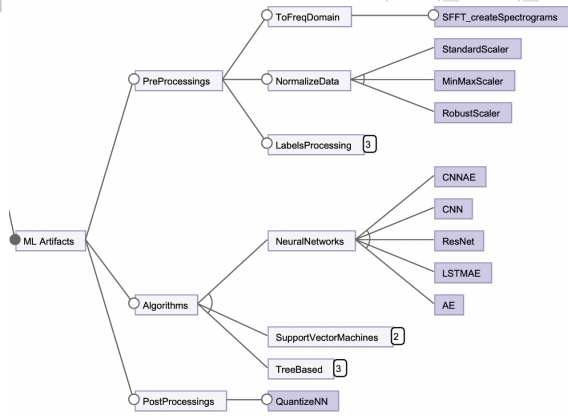
⁶in CPU time, in memory usage, and so on



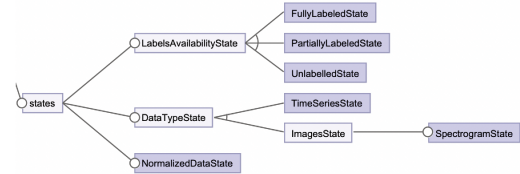
(a) FM on Initial Data



(b) FM on Business Requirements



(c) FM on ML Artifacts

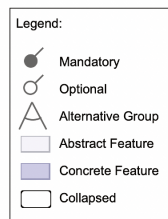


(d) FM on States

- (1) CNNAE → ImagesState ③
- (2) CNN → (ImagesState ∧ FullyLabeledState) ③
- (3) ResNet → (ImagesState ∧ FullyLabeledState) ③
- (4) LSTMAE → TimeSeriesState ③
- (6) CNN → ¬ NovelAnomaliesEmergeInProd ④
- (7) ResNet → ¬ NovelAnomaliesEmergeInProd ④

- (10) QuantizeNN → microcontroller ④
- (11) quad FullyLabeledState → (FullyLabeled ∨ (PartiallyLabeled ∧ augmentation)) ② & ③
- (14) NormalizedDataState → (NormalizedData ∨ (¬ NormalizedData ∧ NormalizeData)) ② & ③

(e) Domain variability model set of constraints



(f) Legend

Figure 6.7: Features and constraints

of data properties that may change over time. For instance, converting data from time series to a collection of spectrograms alters the data type to images, thereby enabling the utilization of solution components that may have been restricted before.

We then limit the space of realized products to two experiments with their associated notebooks. XP1 is an experiment conducted on non-labeled and non-normalized time series data of motor vibrations to detect acquisition errors in the data, specifically the occurrence of single-point anomalies. The associated notebook includes a normalization algorithm (*e.g.*, `MinMaxScaler`) and an LSTM-type neural network. However, due to the non-quantizable nature of LSTM, this experiment cannot be embedded.

XP2, which refers to NB2, pertains to an experiment conducted on partially labeled and non-normalized time series data of sounds to detect identifiable malfunction anomalies in motor sounds with abnormal patterns (*e.g.*, `patternAnomaly`). The notebook associated with this experiment employs a spectrogram calculation to transform data into the time-frequency domain, enabling better detection of anomalies via (*e.g.*, `SFFT_createSpectrograms`), normalization via `MinMaxScaler`, and a CNN-type neural network. This model was not embedded in a microcontroller.

6.4.3.1 Domain variability model

Through collaboration with our industrial partners, we have conducted the domain engineering phase to gain a comprehensive understanding of the company's knowledge. This phase involved identifying the relevant criteria that data scientists consider in use case analysis, such as business requirements and data properties, to ensure thorough tracing in the feature model.

We used 22 features on `InitialData` to characterize the experiments conducted within the company. Of these, 16 are computed by the metadata computer, one is computed on-demand (stationarity), and five are expert-generated questions. This clearly highlights the necessity of working with partial configurations within this space.

We have also identified and characterized approximately 30 features associated with business requirements. At the time of writing 61% of these features have been used, leaving several features yet to be exploited. Notably, limitations in resources have not yet been factored into our analysis, despite being a key goal of the company.

We also referenced in `MLArtifacts` submodel both algorithms from the literature and algorithms proprietary to the enterprise.

Finally, we managed to define more than 20 states that reflect the state of the data at a particular time.

The metrics in table 6.1 show the number of features for subdomains and the leaf coverage rate (Theoretic selection rate) of the feature model. This rate is the number of selected feature leaves divided by the total number of feature leaves multiplied by 100. It represents the percentage of features really used in configurations, showing here that our experiment actually acts upon a relevant number of features.

6.4.3.2 Variability model of realized products and assets

We entirely followed the approach described in section 6.4 to define `Experiment`, `Workflow Products`, and `Legacy Workflow Representations Products` (LWRP).

	$N_{Features}$	N_{leaf}	<i>Coverage</i>
InitialData	33	22	54%
BusinessRequirements	41	31	61%
ML artifacts	109	67	34%
States	35	21	-
Global*	249	160	37.04%

Table 6.1: Metrics on the industrial knowledge base

The implementation of the prototype was finalized enough so that LWRP can be created, saved, modified, cloned, and deployed using the company software.

In our experimental work, we successfully extracted a range of products comprising 20 distinct elements, including 7 intended for industrial use and 13 benchmark use cases. To enable practical reusability, we developed 16 different workflows and accounted for 67 artifacts. In the context of industrial application, we were able to retrieve solution products (equivalent to notebook products) through the MLArtifacts within the LWRPs.

We also characterized and built solutions using the company anomaly detection software with a pool of available ML artifacts to compose a workflow.

6.4.3.3 First results on the usefulness of configurations in determining workflow suitability

We aimed to validate the usefulness of configurations in determining the suitability of notebooks and thus reduce the analysis space of workflows to be reused. To achieve this, we systematically replayed configurations on the 20 available experiments, firstly by configuring the properties of the initial data, secondly by configuring the business requirements, and finally, by configuring both.

XP #id	Initial Data	Business Requirements	Both
1	20	19	4
2	19	20	3
3	20	20	3
4	20	20	2
5	20	20	2
6	20	20	5
7	20	20	2
8	20	20	3
9	19	19	2
10	19	20	5
11	20	20	3
12	20	20	5
13	20	20	4
14	20	20	2
15	20	19	2
16	20	20	5
17	20	20	3
18	16	20	2
19	16	20	5
20	20	20	2

Table 6.2: Number of notebooks after configuration

The results presented in table 6.2⁷ show that configuring the subspaces separately only leads to a minor reduction in the size of the resulting product space, with the maximum reduction being 4 products in the case of the configuration of data properties of experiment number 18. In contrast, configuring both subspaces together significantly reduces the product space by ignoring up to 18 products in 8 experiments and a minimum of 15 products in 4 experiments. These initial results are encouraging and confirm that solely considering the data is insufficient to determine the optimal solution for a machine learning problem. It is also necessary to consider the business requirements and their interactions with the data. Still, further investigation is required to demonstrate that the generative process can effectively assist users in addressing new problems.

6.5 Conclusion

In this chapter, we proposed a first tooled approach to tackle the problem of diversity machine learning workflows. To help data scientists tailor a workflow to their own problem, our approach uses several variability models coupled with a constraint system. This allows the user to get feedback according to the data and business requirements of the new problem. The presented scenarios in the Application section demonstrated that the user can retrieve past experiments based on problem configuration, or can totally tailor a new one. These scenarios illustrate the potential of the proposed solution. Furthermore, we believe that our findings will be soon applicable to various machine learning subdomains within the constricted application of the company, considering a broader evolving domain modeling through incremental acquisition of knowledge.

Contribution 2. *Developing the model of separation of concerns between the steps to develop a contextualized machine learning workflow by highlighting the importance of the characterization of the related ml problems with the data and the business requirements and guiding the data scientist through formalized human insights/knowledge/practices.*

⁷The table shows the count of remaining notebooks or LWR instances after configuration with respect to initial data, business requirements, or both. The initial number of notebooks and LWR instances is 20.

CHAPTER 7

Accommodating the incremental acquisition of knowledge

chap_splevol

7.1	Introduction	81
7.2	Identifying evolution smells to uncover implicit knowledge	82
7.2.1	Formalization	83
7.2.2	Pattern detections	85
7.2.3	Limitations and Future Directions	85
7.3	Knowledge assessment metrics	86
7.3.1	Domain-Level Feature Model Metrics	86
7.3.2	Evaluating Feature Model through the Lens of Produced Products	86
7.3.2.1	Feature-level metrics	86
7.3.2.2	Feature Model Level Metrics	87
7.3.3	Limitations and Future Directions	88
7.4	Evolving Configurations to Align with Feature Model Evolution	88
7.4.1	Application-preserving refactoring against practice evolution	88
7.4.2	Enhancing Knowledge Through the Integration of New Application	89
7.4.3	Limits and Future Directions	89
7.5	Applications	90
7.5.1	First three steps of the SPL construction process	90
7.5.2	Knowledge extraction driven by SPL assessment	91
7.5.3	Knowledge extraction driven by SPL evolution assessment	92
7.5.3.1	Pattern evolution and knowledge consolidation	92
7.5.3.2	Pattern evolution and knowledge extraction	93
7.5.4	Exploiting the metrics	93
7.6	Conclusion	94

7.1 Introduction

As presented in the previous chapter, the SPL we designed establishes a set of well-defined ML practices with both shared aspects and varying elements modeling variability. Each product is identified through a combination of features, possibly not valued (*i.e.*, a configuration), resulting in an `Application product`, encompassing an ML workflow along with its performance, evaluation strategy, authors, initial data properties, and business requirements. The format of the *ML workflows products* associated with the `Applications products` contains ranges of products, from notebooks to references of runs in OpenML or references to solutions developed by our partner company, EZAKO. The principle is simple: Each product’s configuration is valid if it adheres to the SPL’s rules, and invalid if not.

As we have already seen, this product base is used to help identify ready-made solutions. In this chapter, we confront these solutions with the evolution of knowledge in ML and use it to extract new knowledge. Indeed, as stated in **Takeaway 14**, the design of machine learning workflows is prone to evolution as our understanding of the relationships among artifacts evolves to design more sound workflows. Moreover, we cannot accurately forecast how features and rules will transform in the steps ahead due to the unpredictable nature of identifying anomalies in time-based data and the necessity to stay current in a swiftly evolving domain. We must remain adaptable to incorporate new insights derived from real-world experience. Hence, we have designed an approach to streamline configuration adjustments and evaluate the ongoing validity of available configurations from past scenarios. These evolutions translate into adapting the feature model or including fresh features to enable configurations that were not initially feasible or valid. In our field of study, these enrichments are frequent and can even introduce contradictions relative to past configurations.

To address these evolutions while preserving, or even taking advantage of, the richness provided by past products, we have deliberately adopted a series of strategies described in this chapter. Given their particularity, we have chosen to deepen the perspectives of each of these strategies in this chapter, choosing not to summarize them in the conclusion.

In line with detecting patterns of evolution of Feature models stated by **Takeaway 15**, we are interested in identifying configuration patterns dedicated to the ML domain using the modularity brought by our separation into subdomains and their evolutions, such as similar problems with different solutions or the same solutions for different problems (cf. 7.2).

As highlighted in **Takeaway 16**, the ability to analyze and measure the knowledge state within the product line is essential. In this chapter, we introduce metrics that enable us to monitor the evolution of our product line and extract new insights from it (cf. 7.3). We particularly leverage the set of product configurations for this purpose.

To maintain the richness of our products, we have devised a process to align product configurations (cf. 7.4), specifically tailored for ML applications and leveraging the modularity discussed in Chapter 6. These configuration updates also serve as a source of information regarding the knowledge encapsulated within the product line.

We built the line itself following these different principles. We thus explain their applications (cf. 7.5) before concluding. We’ve devised tools to validate the complete process, encompassing (i) configuration and retrieval of previous `Applications products` and ML workflows products (configurator), (ii) ML workflow generation (generator), (iii) incor-

poration of Applications products and ML workflow products into the SPL (integrator), (iv) reconfiguration of past setups (re-configurator), and (v) assessment of the knowledge within the SPL regarding recorded Applications products (analyzer).

Consequently, this chapter is structured as follows: firstly, we explain how we formalize equivalence classes of configurations to detect three bad smells (cf. 7.2); secondly, we introduce metrics for assessing the evolution of knowledge (cf. 7.3); thirdly, we outline our approach for aligning previous configurations with updated knowledge (cf. 7.4); and finally, we demonstrate our system's functionality by showcasing three major stages of the line's evolution, each contributing to the enrichment of knowledge (cf. 7.5). Through this application, we provide insights into the operation of our system.

7.2 Identifying evolution smells to uncover implicit knowledge

Similar to Tornhill (Tornhill, 2015), we seek to identify "hot spots" to narrow our study of Applications products to a few critical cues most likely to guide us in extracting new knowledge. To achieve this objective, we seek to establish a link between the problem description based on data and the business requirements relating to the solution. We assume a direct relationship exists between problem characteristics (Initial data and Business requirements) and corresponding solutions (ML artefacts) in the context of generating ML workflow products for a given machine learning use case or application. This assumption allows us to detect different situations. (i) One of the solutions is not adapted to the problem, and in this case, in retrospect, the data scientist should not have used it. We must enrich the feature model to prohibit it. (ii) The two problems are different, but we had not yet identified these discriminative criteria in the feature model; we must enrich the feature model with these new criteria. To identify these different situations more precisely, we look for different patterns.

P1: Two problems evaluated as equivalent have a different solution. Characterizing this pattern allows us to detect different situations. (i) One of the solutions is not adapted to the problem, and in this case, in retrospect, the data scientist should not have used it. We must enrich the feature model to prohibit it. (ii) The two problems are different, but we still need to identify these discriminative criteria in the feature model; thus, we must enrich it with these new criteria. Because configurations only partially characterize the problems and involve undetermined criteria, we expand our identification process from equivalent problems to unifiable problems.

P2: Two unifiable problems have different solutions. In other words, given the partial information available on the problem, if compatible criteria characterize two problems, it may be the same situation as in the previous case. However, the absence of information may have led the data scientist to design a solution that adapts to the missing information, for example, the absence of information on the types of anomalies, which is also information.

P3: Two solutions evaluated as equivalent solve two non-equivalent problems. This scenario, where several problems have the same solution, can lead to identifying the insensitivity of the solution to specific characteristics. In such cases, certain criteria may be identified as unnecessary or too detailed to discriminate. When problems are unifiable, this issue is considered a weaker warning than when two non-equivalent problems yield the same solution.

To further clarify these patterns, we specify the notions of *equivalence classes* and their *unifiability*.

7.2.1 Formalization

To illustrate the following definitions, we use the feature model presented in Figure 7.1 and the configurations described in Table 7.1. fig_FMmetrics

Let a feature model FM and A a set of valid partial configurations of FM , $A \subseteq \llbracket FM \rrbracket$.

In order to detect the patterns introduced previously, we now define the notion of equivalence classes on a subset of features. We will then apply it to subdomains.

Equivalence class of configurations on a subset of features An equivalence class $[c]$ of configurations defined on a subset of features F corresponds to the set of valid configurations ($c_i \in A$) which share the same selection states for the features of the subset F , *i.e.*, each feature $f_i \in F$ must be selected, deselected, or left undefined in all configurations c_1, \dots, c_k in the equivalence class $[c]$.

Définition 7.2.1 (Equivalence Class on A according to a subset of features of FM). An equivalence class $[c]$ on a subset of features F of FM is defined as a set of valid configurations in A , such as $[c] = \{c_1, \dots, c_k\}$, $c_i \in A$,

$$\forall f_i \in F, f_i \in \bigcap_1^k \text{selected}(c_j) \cup \bigcap_1^k \text{deselected}(c_j) \cup \bigcap_1^k \text{undefined}(c_j).$$

Table 7.2 shows the identified equivalence classes on different subtrees. Two equivalence classes are identified on the `InitialData` subtree. tab_classesOnExample

$[CDS1] = \{\text{app1}, \text{app2}, \text{app4}\}$. $CDS1$ can also be noted: $\{(d1, s), (d2, u), (d3, u)\}$.
 $[CDS2] = \{\text{app3}\}$, $CDS1$: $\{(d1, s), (d2, s), (d3, u)\}$

To identify patterns P1, P2, and P3, we now characterize the equivalence classes on the reduced space of problems (initial data and Business Requirements) and solutions.

Définition 7.2.2 (Problem equivalence classes). *Problem equivalence classes* are defined on the sub-features of `InitialData` and `BusinessRequirements`.

For example, in table 7.2, $[CP1] = \{\text{app1}, \text{app2}, \text{app4}\}$ corresponds to the following feature configurations $CP1_{\{d1, s, d2, u, d3, u, p1, d, p2, s, p3, u, p4, s\}}$ tab_classesOnExample

Définition 7.2.3 (Solution equivalence classes). *Solution equivalence classes* are defined on the sub-features of `ML artefacts`.

Configuration equivalence Two different configurations are equivalent in FM if they are members of the same Problem and Solution equivalence classes. In our example, `app1` and `app2` are equivalent.

Unifiable classes Two different equivalence classes $[c1]$ and $[c2]$, defined on the same set of features F , are considered unifiable if the differences in feature assignment only involve non-selected features.

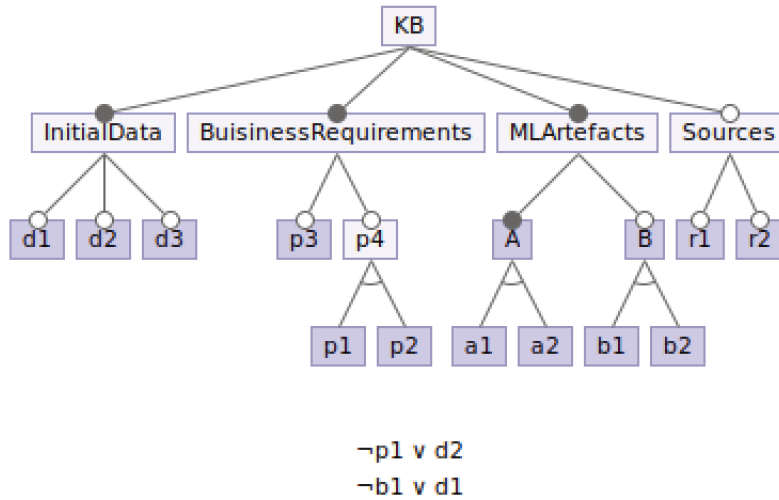


Figure 7.1: Feature model to illustrate definitions and metrics

Table 7.1: Examples of Applications products configurations.

XP Name	d1	d2	d3	p3	p4	p1	p2	a1	a2	b1	b2	Sources	r1	r2
app1	as	u	u	u	as	ad	ms	md	as	ms	md	u	u	u
app2	as	u	u	u	as	ad	ms	ad	ms	ms	md	u	u	u
app3	as	as	u	u	ms	u	ad	ad	ms	ms	md	u	u	u
app4	as	u	u	u	as	ad	ms	ms	ad	ms	md	as	ms	u

s=selected, d=deselected, u=undefined, a=automatic, m>manual

Définition 7.2.4 (Unifiable classes). Two different equivalence classes $[c_1]$ and $[c_2]$ defined on a same set of features F are unifiable if

$$\forall f_i \in F, f_i \in \cap_1^2(\text{selected}(c_j) \cup \text{undefined}(c_j)) \cup \cap_1^2(\text{deselected}(c_j) \cup \text{undefined}(c_j))$$

In the example, in table 7.2, CDS1 and CDS2 are unifiable, and CP1 and CP2 are unifiable too.

Table 7.2: Equivalence classes

App Name	InitialDSCClass	InitialPBClass	SolutionClass	EquivalentApp
app1	CDS1	CP1	CS1	{app2}
app2	CDS1	CP1	CS1	{app1}
app4	CDS1	CP1	CS2	[]
app3	CDS2	CP2	CS1	[]

app1, app2 and app4 handles the same equivalence classes of dataset (CDS1) and problem (CP1).

app1 and app2 are equivalent.

app3 deals with the same equivalence class of solutions (CS1) than app1 and app2).

According to Table 7.1, CP1 and CP2 are unifiable.

Table 7.3: Pattern detection.

App Name	P1	P2	P3 _{weak}
app1	{app4}	{app3}	{app3}
app2	{app4}	[]	{app3}
app4	{app1,app2}	{app3}	{}
app3	{}	{app4}	{app1, app2}

While *app4* handles the same equivalence class of problem as *app1*, it proposes another solution, P1 is detected.

CP1 and *CP2* are unifiable. *app3* and *app4* do not belong to the same equivalence class of solution, P2 is detected.

app3, *app1*, and *app2* define the same equivalent solution but address different unifiable problems; P3 is weakly detected.

7.2.2 Pattern detections

P1 detection For any c_i and c_j in the equivalence class of problems $[cp]$, if they do not belong to the same equivalence class of solutions, we have detected the P1 pattern.

In the example presented in table 7.3, we observe the detection of pattern P1 between *app4* and the applications *app1* and *app2*; they all belong to the same problem equivalence class, CP1 but *app4* does not belong to the CS1 equivalence class of solutions.

P2 detection For any c_i and c_j in two different equivalence classes of problems $[cp_i]$ and $[cp_j]$, but $[cp_i]$ and $[cp_j]$ are unifiable, if c_i and c_j do not belong to the same equivalence class of solutions, we have detected the P2 pattern.

In the example presented in table 7.3, we observe the detection of pattern P2 between *app3* and the application *app4*; they belong to two unifiable problem equivalence class, CP1 and CP2 but *app4* and *app3* belong to two different equivalence classes of solutions.

P3 detection For any c_i and c_j in the equivalence class of solution $[cs]$, if they do not belong to the same equivalence class of problems, we have detected the P3 pattern. However, if the classes of problems are unifiable, P3 is weakly detected.

In the example presented in table 7.3, we observe the detection of pattern P3 between *app3* and the applications *app1* and *app2*; they belong to the same solution equivalence class, CS1, but they belong to different but unifiable problem equivalence classes.

7.2.3 Limitations and Future Directions

In this section, we focused on identifying equivalence classes based on features. Moving forward, we aim to refine this approach to a higher level of abstraction to uncover workflow patterns. For example, we seek to investigate whether all solutions within a problem family belong to the same workflow family, as tools driven by business domains such as the one Azure and IBM propose. The number of applications in an industrial context is low, with each development corresponding to at least six months of work. Consequently, we were able to handle the identified patterns. As we will see in the application section, they have indeed facilitated the detection of new insights, particularly by tracing the evolution in detecting the patterns themselves.

Table 7.4: Metrics related to the FM in fig. 7.1 and its configurations (tab. 7.1)

NoF	20	$NoAP$	4
N_{leaf}	12	Cov	66%
NoC	2	$NoEC$	3
$CTCR$	20%	Com	37,5%

NoF : number of features / $NoAP$: number of applications products/ N_{leaf} : number of leafs / Cov : feature model coverage rate / NoC : number of cross constraints / $NoEC$: number of equivalence classes / $CTCR$: cross tree constraints rate / Com : commonalities rate

7.3 Knowledge assessment metrics

We selected some standard metrics (El-Sharkawy, Yamagishi-Eichler, & Schmid, 2019b) to assess the state of the feature model and, by comparison, its evolution.

7.3.1 Domain-Level Feature Model Metrics

The number of features (NoF) and the number of features without children (N_{leaf}) are classic metrics for assessing the scale of an SPL. Our objective is to analyze the state of the line and its evolution in the different subdomains identified in the previous chapter. We, therefore, analyze these metrics in the different spaces. In our example in Figure 7.1, the number of leaves is twelve, which is four in the ML artefact subtree (Solution Space).

The evolution of the number of cross-constraints (NoC), together with the tree-cross-constraint ratio ($CTCR$)¹, gives a numerical indication of the identified interactions. In our example, four features are involved in constraints, so the $CTCR$ is 20%. Given the partial knowledge of the domain, the theoretical number of possible configurations is not computable and does not offer meaningful insights in our context. Some valid configurations may not correspond to suitable solutions.

Furthermore, this partial knowledge of the domain combined with the very high cost of ML workflow evaluations does not make it possible to test the SPL by generating examples unless it consumes many resources without any assurance of a real gain.

On the other hand, we now propose to evaluate the feature model FM according to the set of valid configurations $A \subseteq \llbracket FM \rrbracket$ which correspond to the Applications products integrated into the SPL.

7.3.2 Evaluating Feature Model through the Lens of Produced Products

7.3.2.1 Feature-level metrics

To understand the usage of the feature model, we now introduce various metrics on features.

¹Number of distinct features involved in cross-tree constraints and divides them through the total number of features in the feature model

The commonality ($Com(f)$) indicates the selection ratio (manual or automatic) of a feature f in A . This ratio identifies the "unused variability" smell (i.e., feature always selected, e.g., $Com(b1) = 1$) (Apel, Batory, Kästner, & Saake, 2016).

The rate of deselection ($Des(f)$) identifies the "unused feature" smell (i.e., feature always deselected, e.g., $Com(b2) = 0$, $Des(b2) = 1$) (Apel et al., 2016).

The rate of undefined occurrences ($Und(f)$) identifies an obscure feature that is not well related to the scope of the SPL (e.g., $Und(d3) = 1$, $Des(d3) = 0$, $Com(d3) = 0$).

We globalize these metrics to all feature model leaves, which, in our case study, characterize practices.

7.3.2.2 Feature Model Level Metrics

Feature Model Coverage (Cov) measures (in percentage) the degree of leaf selections in a set of configurations A . *Feature model coverage rate = number of feature leaves selected in A / number of feature leaves * 100.*

In our example, eight leaves are selected at least once, $Cov = 66\%$. For the Solution subtree, as three leaves were chosen at least one time, $Cov_{Solution} = 75\%$

Feature Model Coverage does not correspond to *t-wise* coverage (Henard et al., 2014). Unlike the latter, it only provides a measure of the feature selection rate in a given set of configurations; it does not allow for assessing the coverage of feature interactions. Nevertheless, it has the advantage of not requiring to compute the number of possible configurations.

The Feature Model Commonality Rate (Com) measures (in percentage) the selection ratio of leaf selections in A .

*Feature model Commonality Rate = number of selection of feature leaves in A / number of feature leaves * $\#A$ * 100*, where $\#A$ denotes the cardinality of the set A . Intuitively, the confidence in the feature model suggestions is proportional to its commonality rate.

In our example, 18 selections of leaves for 4 configurations and 12 leaves, $Com = 37,5\%$. $Com_{Solution} = 50\%$

The number of equivalence classes ($NoEC$) complements the analysis of the use of the feature model. Indeed, the fewer equivalence classes there are, the lower the variability used in the feature model. By performing this analysis by subdomain, we characterize the coverage of subdomains by applications.

Feature Model Automation Rate The automatic feature selection rate ($AutR$) is used to evaluate the help provided by the line. In our example, 6 leaves were automatically selected/deselected (2 in Initial Data space), $AutR = 50\%$, $AutR_{InitialData} = 66,7\%$

*Feature model automation rate = number of feature leaves automatically selected/deselected in configurations / number of feature leaves * 100.*

In the solution space, the higher the leaf selection/deselection rate, the more the line helps in solution construction.

7.3.3 Limitations and Future Directions

These various metrics assess the state of the product line at a given time and its evolution by comparing the changes in these metrics. To more accurately evaluate the validity of these metrics, it is necessary to assess their evolution over time, which we will do in the application section within the context of our product line. Applying these metrics to other product lines remains an interesting subject for further study.

7.4 Evolving Configurations to Align with Feature Model Evolution

The product line evolves through various means, including modifications to the Feature Model and constraints among features, particularly following the analysis of detected patterns (cf. 7.4.1). The construction of new applications classically also contributes to the evolution of the product line, for example, by adding algorithms.

However, in our specific context, it is essential to verify and align the configurations of past applications with the new version of the feature model. Indeed, these configurations are used, on the one hand, for detecting patterns, analyzing the line by the metrics, and, of course, as we saw in Chapter 6, to help data scientists find past applications.

In this section, we present the mechanism for aligning the configurations according to the evolutions of the feature model (cf. 7.4.1); then, we show how the aligned configurations are imported into the product line to be accessible through the configurator (cf. 7.4.2).

7.4.1 Application-preserving refactoring against practice evolution

Evolutions of the feature model may lead to past products (*i.e.*, their related configurations) being detected as conflicting with the current feature model (Apel et al., 2016). To promote a safe evolution, a reconfiguration step is performed on all past configurations. For now, reconfiguring a c_s configuration into a c_t configuration with respect to a new feature model FM consists in

- (i) renaming in c_t some of the features of c_s ,
- (ii) omitting the features that disappeared in FM with a warning if they were selected or deselected in c_s ,
- (iii) adding in c_t the new features of FM whose value is known,
- (iv) copying in c_t the other features, then
- (v) replaying c_t in FM to obtain a new valid configuration or to raise an error in the contrary situation.

Replaying a configuration entails re-evaluating all manually selected or deselected features, starting with the features related to the initial data² then the business requirements, and finally, the

²The automated data analysis process prepares configurations for the initial data part, pre-selects or deselects features, and marks them as manually selected. We deliberately avoid replaying this resource-intensive step unless the data analysis component evolves and the extracted information is necessary for the evolution of the product line. This also raises concerns about data access, which could potentially be problematic.

solution. We consider a conflict to arise in this step only if features can no longer be selected or deselected according to the original configuration due to the imposition of new constraints in the feature model. However, we do not raise a warning if features were automatically selected/deselected. Then, in the same order (initial data set), if features that were previously automatically selected or deselected are not so anymore, we select them manually and note this as a "manual" selection/deselection. At the end of this process, if no conflicts are detected, we analyze the configuration to identify previously unvalued features (either because they were non-existent or unknown) to indicate those that have been automatically valued.

If past configurations cannot be rendered valid in FM , we exclude them with a warning. The new valid configurations related to `Applications products` can then be integrated into the FM .

For example, if we add the constraint $\neg b1 \vee \neg d2$ in FM of Figure 7.1, the configuration corresponding to `app3` is not valid anymore, while all the other configurations are automatically updated with $(d2, ad)$.

singApplications

7.4.2 Enhancing Knowledge Through the Integration of New Application

Integrating in the feature model an `Application product` named `app` on a dataset named `d` and defined by a valid configuration `c` consists in adding, in the `Applications product` branch of the feature model, the features `d` and `app` if they are not already there. Then, the minimal constraints³ linking `d` to the selected and deselected features of the `initialData` space are added starting from the manually selected and unselected leaves. The constraint $app \Rightarrow d$ is then added. We then proceed in the same way to link `app` to the rest of the feature model, starting with the problem space. When the dataset `d` is already present in the feature model, there should be no contradiction with its constraints. However, they can be completed when the same dataset is used in other applications. For example, the data scientist John enters a new application on dataset 2 defined on `app3`, whereas when he entered the data set description he was able to manually inform the criteria `d3` previously unknown by manually selecting it, this operation updates dataset 2 and therefore updates the `ds2` equivalence class at the same time as the dataset is shared by both applications. He kept informing the rest of the problem and the solution he used resulting in `app5` being defined by the following configuration

$\{(d1, ms), (d2, as), (d3, ms), (p1, ms), (p2, ad), (a1, ms), (b1, md), (b2, ms), \dots\}$ then he adds the feature `app5` in the branch `Applications` and the following constraints: $ds2 \Rightarrow d3, app5 \Rightarrow ds2, app5 \Rightarrow p1, app5 \Rightarrow b2 \wedge \neg b1 \wedge a1, app5 \Rightarrow John$ This application addition also introduced a new class of problem and solution, enriching our knowledge while raising the identification of a P2-type problem.

7.4.3 Limits and Future Directions

The alignment processes have been implemented and utilized in the construction of the SPL. These different steps are described in the following section. While conflict detection during alignment is operational, it remains challenging to interpret when the number of features is large, and the FM modifications are numerous. Dedicated visualizations should be explored to facilitate this crucial and time-consuming analysis. Nevertheless, it is worth noting that if we consider this alignment step for all past configurations as regression tests, we have never detected any regressions at this

³features automatically selected or deselected during the configuration are not involved in new constraints

stage. Analyzing the modification of past configurations opens avenues to address different challenges. For example, the evolution of metrics such as coverage rates or automation can qualify changes and drive new space explorations. Similarly, the evolution of detected patterns (unifiable problems becoming equivalent or, conversely, different) is another aspect we aim to analyze formally.

7.5 Applications

We now report on the first three steps of the SPL's construction, showing how the practices contributed to its enrichment. Figure 7.2 summarizes this construction process. The configurations and the results of the analyses are accessible online⁴.

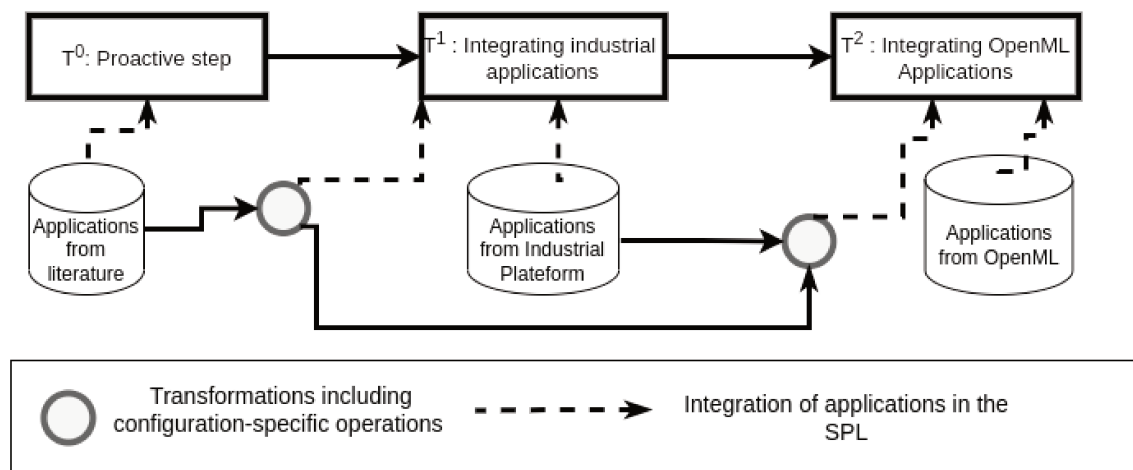


Figure 7.2: Three main steps of our SPL construction

7.5.1 First three steps of the SPL construction process

In each of the steps presented below, we have integrated the `Applications` products into the FM.

ss_T0 Initial product line version from literature stud Following a first analysis of the domain, we built the SPL's initial version (SPL_{T^0}). The feature model (FM_{T^0}) integrates some solutions from the literature for detecting time series anomalies. The `Applications` products correspond to experiments carried out on reference data sets (Dau et al., 2018).

ss_UPALGO Enrichment of the product line through industrial practice At T^1 , we leverage the practices of the partner company's data scientists to build SPL_{T^1} by enriching SPL_{T^0} . The interest in exploiting industrial `Applications` products is to broaden the scope of the SPL to the processing of industrial data. The industrial partner uses a custom tool to summarize all the `Applications` products on their customer's data. We update the feature model

⁴<https://anonymous.4open.science/r/RFTS-SPLC2022-D508/>

($FM_{T^0} \rightarrow FM_{T^1}$) by including company-specific solution components, new initial dataset properties relevant to analyzing customer datasets, and new features necessary to describe the customer business requirements. Then we collect `Applications products` conducted by the company’s data scientists, keeping only the solutions from deployed workflows and solving customers’ anomaly detection problems. We have thus selected six workflows whose resulting product models are in production. The construction of these workflows can take several months for the data scientists. We have generated partial configurations containing information about the dataset and solution based on automatic solution extraction and data analysis tools. We used these partial configurations to initialize the configurator. We then completed the source data and business requirements parts via a discussion with the `Application product` authors.

ss_OpenML

Consolidation by extraction of OpenML workflow At T^2 , we extract some practices from the OpenML platform. OpenML is an automated machine learning environment (Vanschoren et al., 2013), from which ML practices can be downloaded and uploaded *i.e.*, solutions (runs and flows in OpenML) to a given problem (task and dataset in OpenML). The interest in exploiting OpenML’s practices is to analyze the impact of upgrading the SPL with external sources. In OpenML, we selected time-series datasets and associated tasks of type *Supervised Learning* and *Unsupervised learning* since anomaly detection is supervised or unsupervised learning with unbalanced classes. We only had four datasets that matched these criteria. We kept 4 tasks of *Supervised learning* that had runs associated with them. Among these runs, we selected only the best runs on F1-score evaluation criterion as evaluations on other measures such as user CPU-time were not available for these runs. We preferred the runs using the scikitlearn library when we had the choice. We then extracted the associated flows and generated the associated partial configurations for each run. We had already studied in T^0 the meta-features proposed by OpenML to characterize datasets, so we only updated the feature model($FM_{T^1} \rightarrow FM_{T^2}$) by adding new solution components.

kextraction

7.5.2 Knowledge extraction driven by SPL assessment

We explain in the following subsections how we exploit pattern and metric analysis in our use case.

Two different solutions for the same problem: Algorithms side effects At T^0 , we encountered the following scenario. For two equivalent problems, the solutions used two different scaling techniques in each workflow, min-max scaler and robust scaler (Patro & Sahu, 2015). This equivalence of problems and not solutions raised a P1 warning. We analyzed workflows for both experiments and observed that the robust scaler results were equivalent to the min-max scaler results for the second workflow due to the data properties. In this scenario, we could confirm that the robust scaler’s main particularity was not required⁵. Therefore, only the first application with min-max scaler was kept. We added a constraint to the selection of this algorithm to prevent the error from being repeated. *i.e.*, data without outliers will no anymore be scaled with a robust scaler.

Two different solutions for the same problem: Data Scientist preferences impact The data scientist’s preferences bias her choice of the solution components. At T^1 , we identified two applications that presented different solutions to equivalent problems (P1). The two authors could not

⁵Usage of the robust scaler is interesting only if outliers are within the values of the time series

justify the difference in the choice of Solution components other than by their expertise in selected algorithms. Therefore, we have kept these two applications distinguishable by their author, with a warning for possible future treatment.

Two problems same solution: Factorizing unnecessary variability At T^0 , two problems differ only in acquisition sampling; data acquisition sampling is in seconds for one and in microseconds for the second. Otherwise, the data are similar, and the anomaly detection requirements are equivalent. After detecting this pattern (P3), we checked the impact of acquisition sampling on the algorithms and factorized all four regular sampling features into `regularSampling` for the SPL at T^1 .

As indicated in this section, the analyses carried out at this stage provide valuable information but do not remove all bad smells. We propose conducting additional analyses that consider temporal developments and changes over time to improve this analysis.

Table 7.5: Metrics Evolution in times and spaces

Time steps	sub spaces	<i>NoF</i>	<i>Nleaf</i>	<i>Cov</i>	<i>Com</i>	<i>NoEC</i>	<i>NoAP</i>	<i>NoC</i>	<i>CTCR</i>
T^0	InitialData	23	16	37%	19%	5	-	-	-
	BusinessRequirements	33	24	41%	21,25 %	7	-	-	-
	ML artifacts	51	25	52%	16.8 %	7	-	-	-
	Global*	156	96	35,04 %	14,68 %	10	10	25	21,19%
T^1	InitialData	28	19	42,10 %	18,94 %	9	-	-	-
	BusinessRequirements	43	33	54,55 %	17 %	13	-	-	-
	ML artifacts	67	37	48,64 %	8,64 %	11	-	-	-
	Global*	194	124	40,32 %	10,86 %	14	15	31	21,76%
T^2	InitialData	28	19	57,9 %	18,00 %	14	-	-	-
	BusinessRequirements	43	33	57,6 %	17,24 %	17	-	-	-
	ML artifacts	74	42	57,14 %	7,89 %	15	-	-	-
	Global*	203	131	47,32 %	10,41 %	18	19	32	21,78%

*The difference between the global figures and the figures of the 3 spaces corresponds to the branches Sources and states.

The feature model hierarchy is six levels deep for the Solution branch, and four for the InitialData and BusinessRequirements branches.

7.5.3 Knowledge extraction driven by SPL evolution assessment

We also exploited the analysis of the evolution of patterns and metrics as another source of information.

7.5.3.1 Pattern evolution and knowledge consolidation

At T^0 , we detected P1 pattern on two applications. While they belong to the same class of problem, they are solved by two different clustering models⁶, *kmeans* (Huang et al., 2016) on the one hand and *Dbscan* (Schubert, Sander, Ester, Kriegl, & Xu, 2017) on the other. At T^0 , we did not know which to delete; we kept both configurations.

⁶Solution workflows vary according to machine learning algorithms

At T^1 , we reconfigured the configurations to align with the new feature model, which now incorporates features detailing business expert insights into possible outliers in the data⁷. The feature model also includes associated constraints expressing compatibility between solution components and these new features.

The configuration alignment made it possible to distinguish the two problems and the adequacy of the two different solutions.

7.5.3.2 Pattern evolution and knowledge extraction

At T^0 , we detected P1 pattern on two applications. While they belong to the same class of problem, one α includes a dimension reduction process through PCA (Abdi & Williams, 2010) while the other β skips this step. Like in the previous example, we kept both configurations.

At T^1 , we extended the `InitialData` space with features to explicit time series dimensionalities and automated their evaluation by dataset analysis. The reconfiguration step indicated that the time series were multivariate in α . In contrast, β 's time series were uni-variate (Aboagye-Sarfo et al., 2015). This unique change in configuration highlighted the link between *PCA* and *time series dimensionalities*; we added this constraint in the feature model.

7.5.4 Exploiting the metrics

The metrics defined in section [7.3](#) ^{Is Metrics} help us assess the evolution of the practices in each space.

InitialData The coverage rate (*Cov*) increased from T^0 to T^1 , while the number of features (*NoF*) also increased. This increase indicates that the industrial `Applications products` cover different data set properties from the first `Applications products` on benchmark datasets. Between T^1 and T^2 the coverage increased while the number of features did not change. New `Applications products` did involve new features of the `InitialData`.

We rely on commonality analysis to better understand the variations between industrial and benchmark datasets. This rate made it possible to identify at T^0 that the features relating to `Missingvalues` were always deselected; All datasets had no missing values of any type. We knew we had to cover this situation. At T^1 `MCARMV`⁸, and `StructuralMV`⁹ features had a $com(f) > 1$, which means that the new datasets were exhibiting these two types of missing values. Similarly, we identify irregular sampling time series emergence at T^1 .

BusinessRequirements Within this feature space, we sought to identify the questions experts answered the least. These questions may need rephrasing. The principle is then to identify the most undefined features of the penultimate level. We have not yet met such a case, which the data scientists confirmed.

The coverage and commonality analysis highlights the requirements of industrial `Applications products` for memory, CPU, or energy consumption optimization. The features representing these hardware constraints are either undefined or deselected at T^0 and T^2 . They are selected at T^1 only.

⁷The data scientists can decide whether outliers are anomalies in the context of the experiment

⁸Missing value completely at random

⁹Missing values of structural nature

ML artifacts The coverage rate decreases at step T^1 and increases at T^2 , while the number of features increases strictly. The evolution of these two metrics indicates: (i) on the one hand, that industrial `Applications` products use new solution components; (ii) and on the other hand, that the `Applications` products we integrated at T^2 consolidate our SPL by reusing existing solution components. The commonality rate decreases to reach 7.89%. However, a detailed analysis of the number of selections by feature indicates that some algorithms are used in several solutions while others are never used. For instance, we observe that each of `LSTMAE` (LSTM Auto-encoder) and `MAE` (mean absolute error) have been used 5 times out of 19, while `padding`, `FrontFill` and others have not been used. Therefore, correlated with broader coverage of problem space, this metric should help identify some of the preferences of data scientists and maybe some bias. Indeed, it is natural to think that data scientists generally rely on the algorithms they are comfortable with, sometimes maybe at the expense of the solution.

7.6 Conclusion

In this chapter, we have proposed a practice-driven approach to building an SPL as a first step toward allowing the design of generic solutions to detect anomalies in time series while capturing new knowledge and capitalizing on the existing one.

The incrementality in the acquisition of knowledge and the instability of the domain (Niehaus et al., 2005) *everything changes ... in an unpredictable way* are supported by the SPL through its structuring and the exploitation of partial configurations associated with past `Applications` products. As far as we know, this is the first case of application of the SPL paradigm in such a context and with a knowledge acquisition objective. By capturing practices in partial configurations, we obtain the abstractions to reason about datasets, solutions, and business requirements. The SPL is then used to produce new solutions, compare them to past solutions, and identify knowledge that was not explicit. The growing abstraction supported by the SPL also brings other benefits. In knowledge sharing, we have observed a shift in the approach to creating ML workflows, focusing on analyzing problems before looking for similar applications, especially in choosing evaluation metrics. It is rather difficult for data scientists to explain the precise reasons for their choice. Focusing only on particular cases identified as patterns makes the relevant criteria explicit.

Contribution 3. *Accommodating the evolution of machine learning effective knowledge and assessing it through SPL's mechanisms and metrics. Plus, the detection of incoherence between experiments in order to further analyze it and pinpoint where actions should be taken.*

Conclusion & perspectives

Conclusion and Perspectives

8.1 Conclusion

When data scientists need to create machine learning workflows to solve a problem, they first understand the business needs, analyze the data, and then experiment to find a solution. They judge the success of each attempt using metrics like accuracy, recall, and F-score. If these metrics meet expectations on the test data, it's a success; otherwise, it's considered a failure. However, they often don't pinpoint why a workflow fails before trying a new one. This trial-and-error process can involve many attempts because it's not guided and relies on the preferences and knowledge of the data scientist. This intuitive method leads to varying trial counts among data scientists. Also, evaluating solutions on a test set doesn't guarantee performance on real-world data. So, when models are deployed, additional monitoring is needed. If a workflow performs poorly, the whole process might need restarting with adjustments based on new data. Furthermore, each data scientist learns from their own experiences without sharing knowledge. This lack of collaboration can lead to repeated mistakes and oversights. Additionally, the interpretation of similarity between use cases can vary among practitioners, making the process even more subjective. Overall, the process lacks structure and heavily depends on the individual knowledge and decisions of the data scientists involved.

In this work, we present how to mutualize/standardize data science knowledge to help data scientists generate machine learning workflows by guiding them along the phases of the process *i.e.*, inquiring about business requirements, analyzing the data, and reducing the solution space of possible ML artefacts or establishing similarities with past workflows to reuse them. To this aim, we have proposed three main contributions to this problem:

Contribution 1: Integrating Data, Business Requirements, and Solution Components in ML Workflow design.

While automatic approaches focus on data, our approach considers the dependencies between the data, the business requirements, and the solution components. This holistic approach ensures a more comprehensive understanding of the problem and guides the development of appropriate solutions.

Contribution 2: Customizing Workflows for Tailored Solutions by Leveraging Partial and Modular Configurations. Our approach aims to assist data scientists in customizing workflows for their specific problems. We achieve this by employing various variability models and a con-

straint system. This setup enables users to receive feedback based on their data and business requirements, possibly only partially identified.

Additionally, we showed that users can access previous experiments based on problem settings or create entirely new ones.

Contribution 3: Enhancing Software Product Lines Knowledge through New Product Exploitation. We have proposed a practice-driven approach to building an SPL as a first step toward designing generic solutions to detect anomalies in time series, capturing new knowledge and leveraging existing knowledge when dealing with new experiments or use cases. The incremental acquisition of knowledge and the instability of the domain (Niehaus et al., 2005) *everything changes ... in an unpredictable way* are supported by the SPL through its structuring and the exploitation of partial configurations associated with past use cases.

As far as we know, this is the first application of the SPL paradigm in such a context with a knowledge acquisition objective. By capturing practices in partial descriptions of the problems and solutions implemented, we obtain the abstractions to reason about datasets, solutions, and business requirements. The SPL is then used to produce new solutions, compare them to past solutions, and identify previously implicit knowledge.

The growing abstraction supported by the SPL also brings other benefits. In knowledge sharing, we have observed a shift in the approach to creating ML workflows, focusing on analyzing problems before looking for similar applications.

In this work, We guide the process of understanding the machine learning problem by considering the business requirements from the outset, thereby reducing the solution space accordingly. However, we focus on the feature level (features from the feature model). Can we systematically translate these features into natural language questions to facilitate the interview with the expert by prompting a large language model (LLM) to do the translation?

Additionally, we focus only on time series anomaly detection. However, our approach could potentially be extended to other machine learning subtasks, such as tabular data classification, image classification, text summarization, and more, provided that the separation of machine learning concerns related to each subtask is appropriately addressed, But can we extend it to other non-functional properties such as FATE (Fairness, Accountability, Transparency, and Ethics)?

8.2 Perspectives

8.2.1 FATE

sec_fate

In this perspective, we aim to examine the complexities that data scientists face in making decisions that achieve high model performance and align with Fairness, Accountability, Transparency, and Ethics (FATE) criteria. These criteria are becoming increasingly crucial as Artificial Intelligence (AI) and machine learning models are integrated into various aspects of society, from healthcare to finance to criminal justice. Ensuring that these models are fair, accountable, transparent, and ethical is crucial for protecting the rights and interests of individuals and society. On an industrial level, it is crucial to account for the empirical parameters that played a role in the design of the solution and justify that it answers the business requirements. As data scientists, relying solely on high model performance to justify decisions is insufficient. Understanding and complying with norms and practices when implementing AI solutions is essential.

For example, ensuring **Fairness** in a model means that it should not be biased either by the collected data, the annotations, or the model itself. However, identifying a bias in a study target remains difficult. The bias seems evident to us only when we know it and therefore can actively look for it. It is, therefore, essential to be fair in checking that known types of biases are not present and to avoid reproducing past biases.

Accountability is another essential aspect that means the data scientists should be able to explain and justify their decisions, and mechanisms should be in place to assess their accountability and hold them to it if something goes wrong.

Transparency is also crucial, as it enables stakeholders to understand how a model works and what factors it considers when making decisions. This helps build trust and confidence in the machine-learning model or workflow.

Ethics is another key aspect to consider. As AI models are increasingly integrated into various aspects of society, it is vital to consider the ethical implications of their use. For example, suppose a model is used to make healthcare decisions. In that case, it is vital to ensure that it does not perpetuate existing disparities or cause harm to specific groups of individuals. Additionally, data scientists should consider the broader ethical implications of the data they use and how it is collected, stored, and used. Furthermore, it is essential to track the steps of the process and understand who collected the data, prepared it, analyzed the problem, and so on. Therefore, tracking these steps helps to justify the use of algorithms and understand all the steps of the process and how they contribute to the final decision.

These challenges require capitalization of knowledge, systematization of code productions, and guidance of verifications similar to what we presented in the main body of this work. We propose using software product lines (SPL) to meet these challenges. From the requirements relating to the case studies, the objective is (i) to systematically produce suitable ML systems and accurate verifications to be carried out (fairness), (ii) to report the SPL and human choices in line with the state of knowledge (accountability), (iii) to register the final solution, including the justification for the parts that involve the human (transparency).

A feature model can capitalize on knowledge about ML application requirements such as FATE, business, and data-related criteria. By modeling FATE properties (see fig. 8.2 in feature models, we can help data scientists ensure that their models bring value to the stakeholders while being fair, accountable, transparent, and ethical concerning a set baseline and help push that baseline forward when they can by making the FATE branch evolve accordingly to the principles introduced in chapter 7 of the main body of this work.

In summary, we aim to explore the challenges faced by data scientists in ensuring that their decisions align with multiple criteria and understanding the problem they are trying to solve. Here after in [UNDEFINED] is an example of a set of guidelines to evaluate FATE properties when faced with an ML use case according to the modeled features in fig. 8.2

Fairness guidelines with examples applied to ECG Time-Series Classification with Machine Learning:

Data Validation:

- Check for any notable disparities in the distribution of the dataset classes for different subgroups.

Examples of disparities :

- If there are more recordings of ECG data from men compared to women, it might indicate potential gender bias in the data
- If the data has more ECG recordings with a normal heart rate compared to recordings with arrhythmias, the model may not generalize well to cases of arrhythmias. (Representation)

- Ensure the data collected covers the space you target.

Example of space coverages:

- individuals from different races, genders, ages, and other demographic categories. for example, if the target population is patients with heart disease, the data should contain ECG recordings from heart disease patients.

- Check with the domain expert that no protected features are used.

Example of protected features:

- The hospital where the ECG was collected or the practitioner who collected it

Annotation Validation:

- Verify that the annotations used to train the model are consistent and unbiased.

Example annotation validation techniques:

- ECG recordings annotated as "normal" should consistently be labeled as normal across all annotations. (cross annotations)
- Assess the agreement between the annotators to ensure the annotations are reliable. For instance, if most annotators agree that a particular ECG recording is "abnormal," it is likely to be accurate. (cross annotators).

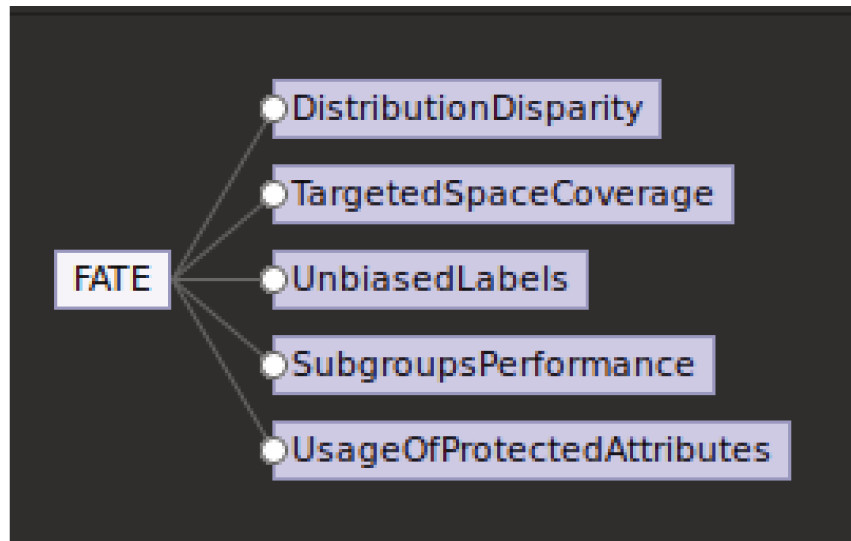
Model Validation:

- Verify that the model's performances are acceptable for all sub-groups of the dataset.

Example of model validation:

- If the data contains ECG recordings from individuals of different ethnicities, it is vital to ensure that the model performance is not underperforming for a particular ethnicity.

Figure 8.1: Fate properties form



fig_FATE

Figure 8.2: Fate properties branch

8.2.2 Using LLMs to conduct the domain expert interview

In our work, one of the limitations was that the domain experts needed the data scientists to explain further the questions and options from the form generated from the feature model, which means that both the domain expert and data scientist needed to be available to retrieve the business requirements. However, we believe this limitation could be effectively bypassed if specific LLMs were developed to conduct the interview with the domain experts, given that they are properly trained and prompted. We have tried GPT 3 and got encouraging results by prompting it, as shown in the sequence of Figures 8.3 to 8.6.

- You** lets play a game, I will give you an xml file that represents a feature model and you will ask me the questions accordingly to the features just before leafs in it. you will ask the questions one by one and will give the options to chose from, you will also provide explanations if asked and accept that the player doesn't know an answer. at the end of the game provide a summary of the answers.
- < 6/6 >
- ChatGPT** That sounds like a fun game! Please go ahead and share the XML file representing the feature model, and we'll get started with the questions.

fig_rules

Figure 8.3: Prompting GPT3 to read configurations and ask questions


- Do you anticipate deploying this system on a special platform?
 - a) Yes, microcontroller
 - b) No

 **You**
a

 **ChatGPT**

1. **eventualAnomaliesInProduction:**

- What types of anomalies do you anticipate encountering in production?
 - a) Data Distribution Shift
 - b) Slow Data Drift

 **You**
i don't know

 **ChatGPT**

No worries! Let's move on to the next question.

Figure 8.4: Answering the chat questions

1. **AnomalyType:**

- What type of anomalies are you expecting to detect?
 - a) Point Anomaly
 - b) Pattern Anomaly
 - c) Global Anomaly



You

what does each of the options mean



ChatGPT

Of course! Here's a brief explanation of each option:

- **Point Anomaly:** Refers to individual data points that are considered anomalous or different from the majority of the data.
- **Pattern Anomaly:** Involves detecting anomalies based on unusual patterns or trends in the data, rather than individual points.
- **Global Anomaly:** Indicates anomalies that affect the entire dataset or a significant portion of it, rather than specific points or patterns.



You

then global anomalies



fig_answers

Figure 8.5: Asking the chat further explanations of the options on anomaly types

1. **LearningMode:**

- Which learning mode would you prefer for anomaly detection?
 - a) Online
 - b) Offline
 - c) Federated



You

what's the difference between the 3 options



ChatGPT

Certainly!

- **Online Learning:** In this mode, the model learns continuously as new data becomes available. It updates its parameters incrementally based on each new data point or small batches of data. This mode is suitable for scenarios where data streams in real-time and the model needs to adapt quickly to changing conditions.
- **Offline Learning:** Also known as batch learning, this mode involves training the model on a fixed dataset without any updates during the learning process. The model is trained on the entire dataset at once, and then its parameters are fixed. This mode is typically used when all the data is available upfront or when periodic retraining is feasible.
- **Federated Learning:** This mode involves training models across multiple decentralized devices or servers holding local data samples, without exchanging them. Each device or server independently computes an update to the global model based on its local data and shares only the model updates, not the data itself. Federated learning is useful when data privacy concerns prohibit centralized data collection, but collaborative model training is still desired.



You

offline then



1. **InitialProblem:** Special deploy platform: Microcontroller
2. **eventualAnomaliesInProduction:** Anomaly types: (No answer provided)
3. **LearningMode:** Offline
4. **LabelsAvailability:** Labels available: Fully Labeled
5. **DataType:** Type of data: Time Series
6. **TimeSeriesProperties:** Properties of time series data: Stationary
7. **TsSampling:** Sampling of time series data: Regular

fig_summary

Figure 8.6: Summary of provided answers

References

- Abdi, H., & Williams, L. J. (2010). Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4), 433–459.
- Aboagye-Sarfo, P., Mai, Q., Sanfilippo, F. M., Preen, D. B., Stewart, L. M., & Fatovich, D. M. (2015). A comparison of multivariate and univariate time series approaches to modelling and forecasting emergency department demand in western australia. *Journal of biomedical informatics*, 57, 62–73.
- Acher, M., Collet, P., Gaignard, A., Lahire, P., Montagnat, J., & France, R. B. (2012). Composing multiple variability artifacts to assemble coherent workflows. *Software Quality Journal*, 20, 689–734.
- Acher, M., Collet, P., Lahire, P., & France, R. (2009). Composing feature models. In *International conference on software language engineering* (pp. 62–81).
- Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., ... Zimmermann, T. (2019). Software Engineering for Machine Learning: A Case Study. In *Proceedings - 2019 IEEE/ACM 41st international conference on software engineering: Software engineering in practice, ICSE-SEIP 2019* (pp. 291–300). Montreal Quebec Canada: IEEE. doi: 10.1109/ICSE-SEIP.2019.00042
- Amraoui, Y. E., Blay-Fornarino, M., Collet, P., Precioso, F., & Muller, J. (2022). Evolvable spl management with partial knowledge: an application to anomaly detection in time series. In *Proceedings of the 26th ACM international systems and software product line conference-volume a* (pp. 222–233).
- Apel, S., Batory, D., Kästner, C., & Saake, G. (2016). *Feature-oriented software product lines*. Springer. Retrieved from <https://link.springer.com/content/pdf/10.1007/978-3-642-37521-7.pdf>
- Arora, R. (2012). Comparative analysis of classification algorithms on different datasets using weka. *International Journal of Computer Applications*, 54(13).
- Babar, M. A., Chen, L., & Shull, F. (2010). Managing variability in software product lines. *IEEE software*, 27(3), 89–91.
- Bagheri, E., Ensan, F., & Gasevic, D. (2012). Decision support for the software product line domain engineering lifecycle. *Automated Software Engineering*, 19, 335–377.
- Bagheri, E., & Gasevic, D. (2011). Assessing the maintainability of software product line feature models using structural metrics. *Software Quality Journal*, 19, 579–612.
- Benavides, D., Segura, S., & Ruiz-Cortés, A. (2010). Automated analysis of feature models 20 years later: A literature review. *Information systems*, 35(6), 615–636.

- Berger, T., & Guo, J. (2014). Towards system analysis with variability model metrics. In *Proceedings of the 8th international workshop on variability modelling of software-intensive systems* (pp. 1–8).
- Berger, T., She, S., Lotufo, R., Wasowski, A., & Czarnecki, K. (2013). A study of variability models and languages in the systems software domain. *IEEE Transactions on Software Engineering*, 39(12), 1611–1640.
- Bergmeir, C. N., Benítez Sánchez, J. M., et al. (2012). Neural networks in r using the stuttgart neural network simulator: Rsnns..
- Berthold, M. R., Cebron, N., Dill, F., Gabriel, T. R., Kötter, T., Meinel, T., ... Wiswedel, B. (2009). Knime-the konstanz information miner: version 2.0 and beyond. *AcM SIGKDD explorations Newsletter*, 11(1), 26–31.
- Bezerra, C. I., Andrade, R. M., & Monteiro, J. M. S. (2014). Measures for quality evaluation of feature models. In *Software reuse for dynamic systems in the cloud and beyond: 14th international conference on software reuse, icsr 2015, miami, fl, usa, january 4-6, 2015. proceedings 14* (pp. 282–297).
- Bezerra, C. I. M., Monteiro, J. M., Andrade, R. M. C., & Rocha, L. S. (2016). Analyzing the feature models maintainability over their evolution process: An exploratory study. In *Proceedings of the 10th international workshop on variability modelling of software-intensive systems* (p. 17–24). New York, NY, USA: Association for Computing Machinery. Retrieved from <https://doi.org/10.1145/2866614.2866617> doi: 10.1145/2866614.2866617
- Bilalli, B., Abelló, A., & Aluja-Banet, T. (2017). On the predictive power of meta-features in OpenML. *International Journal of Applied Mathematics and Computer Science*, 27(4), 697—712.
- Boehmke, B., & Greenwell, B. M. (2019). *Hands-on machine learning with r*. CRC press.
- Brachman, R. J. (1996). The process of knowledge discovery in databases. *Advances in knowledge discovery and data mining*, 37–57.
- Brownlee, J. (2020). *Data preparation for machine learning: data cleaning, feature selection, and data transforms in python*. Machine Learning Mastery.
- Brunton, S. L., Noack, B. R., & Koumoutsakos, P. (2020). Machine learning for fluid mechanics. *Annual review of fluid mechanics*, 52, 477–508.
- Camillieri, C., Parisi, L., Blay-Fornarino, M., Precioso, F., Riveill, M., & Cancela-Vaz, J. (2016). Towards a software product line for machine learning workflows: Focus on supporting evolution. In *10th workshop on models and evolution co-located with acm/ieee 19th international conference on model driven engineering languages and systems (models 2016)*.
- Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3), 1–58.

- Chung, L., & do Prado Leite, J. C. S. (2009). On non-functional requirements in software engineering. *Conceptual modeling: Foundations and applications: Essays in honor of john mylopoulos*, 363–379.
- Chung, L., Nixon, B. A., Yu, E., & Mylopoulos, J. (2012). *Non-functional requirements in software engineering* (Vol. 5). Springer Science & Business Media.
- Clements, P., & Northrop, L. (2002). *Software product lines*. Addison-Wesley Boston.
- Czarnecki, K., Helsen, S., & Eisenecker, U. (2005). Formalizing cardinality-based feature models and their specialization. *Software process: Improvement and practice*, 10(1), 7–29.
- Dau, H. A., Bagnall, A., Kamgar, K., Yeh, C.-C. M., Zhu, Y., Gharghabi, S., ... Keogh, E. (2019). The ucr time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6(6), 1293–1305.
- Dau, H. A., Bagnall, A. J., Kamgar, K., Yeh, C.-C. M., Zhu, Y., Gharghabi, S., ... Keogh, E. J. (2018). The UCR Time Series Archive. *CoRR*, abs/1810.07758, 1–12. Retrieved from <http://arxiv.org/abs/1810.07758>
- Dhungana, D., Seichter, D., Botterweck, G., Rabiser, R., Grunbacher, P., Benavides, D., & Galindo, J. A. (2011). Configuration of multi product lines by bridging heterogeneous variability modeling approaches. In *2011 15th international software product line conference* (pp. 120–129).
- Dubinsky, Y., Rubin, J., Berger, T., Duszynski, S., Becker, M., & Czarnecki, K. (2013). An exploratory study of cloning in industrial software product lines. In *Proceedings of the european conference on software maintenance and reengineering, csmr* (pp. 25–34). Genova, Italy: IEEE. doi: 10.1109/CSMR.2013.13
- Dwivedi, S., Kasliwal, P., & Soni, S. (2016). Comprehensive study of data analytics tools (rapidminer, weka, r tool, knime). In *2016 symposium on colossal data analysis and networking (cdan)* (pp. 1–8).
- El-Sharkawy, S., Yamagishi-Eichler, N., & Schmid, K. (2019a). Metrics for analyzing variability and its implementation in software product lines: A systematic literature review. *Information and Software Technology*, 106, 1–30.
- El-Sharkawy, S., Yamagishi-Eichler, N., & Schmid, K. (2019b, feb). Metrics for analyzing variability and its implementation in software product lines: A systematic literature review. *Information and Software Technology*, 106, 1–30. doi: 10.1016/j.infsof.2018.08.015
- Epperson, W., Wang, A. Y., DeLine, R., & Drucker, S. M. (2022). Strategies for reuse and sharing among data scientists in software teams. In *Proceedings of the 44th international conference on software engineering: Software engineering in practice* (pp. 243–252).
- Fernández-Delgado, M., Cernadas, E., Barro, S., & Amorim, D. (2014). Do we need hundreds of classifiers to solve real world classification problems? *The journal of machine learning research*, 15(1), 3133–3181.

- Fernández-Delgado, M., Cernadas, E., Barro, S., & Amorim, D. (2014, January). Do we need hundreds of classifiers to solve real world classification problems? *The Journal of Machine Learning Research*, 15(1), 3133–3181. Retrieved from <https://jmlr.org/papers/v15/delgado14a.html>
- Fillbrunn, A., Dietz, C., Pfeuffer, J., Rahn, R., Landrum, G. A., & Berthold, M. R. (2017). Knime for reproducible cross-domain analysis of life science data. *Journal of biotechnology*, 261, 149–156.
- Georgakopoulos, D., Hornick, M., & Sheth, A. (1995). An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and parallel Databases*, 3, 119–153.
- Gil, Y., Deelman, E., Ellisman, M., Fahringer, T., Fox, G., Gannon, D., ... Myers, J. (2007). Examining the challenges of scientific workflows. *Computer*, 40(12), 24–32.
- Glinz, M. (2007). On non-functional requirements. In *15th ieee international requirements engineering conference (re 2007)* (pp. 21–26).
- Gnanambal, S., Thangaraj, M., Meenatchi, V., & Gayathri, V. (2018). Classification algorithms with attribute selection: an evaluation study using weka. *International Journal of Advanced Networking and Applications*, 9(6), 3640–3644.
- Gong, Z., Zhong, P., & Hu, W. (2019). Diversity in machine learning. *Ieee Access*, 7, 64323–64350.
- Habibullah, K. M., Gay, G., & Horkoff, J. (2023). Non-functional requirements for machine learning: Understanding current use and challenges among practitioners. *Requirements Engineering*, 28(2), 283–316.
- HaddadPajouh, H., Dehghantanha, A., Khayami, R., & Choo, K.-K. R. (2018). A deep recurrent neural network based approach for internet of things malware threat hunting. *Future Generation Computer Systems*, 85, 88–96.
- Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1), 10–18.
- He, X., Zhao, K., & Chu, X. (2021). Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212, 106622.
- Henard, C., Papadakis, M., Perrouin, G., Klein, J., Heymans, P., & Traon, Y. L. (2014). Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines. *IEEE Transactions on Software Engineering*, 40(7), 650–670. Retrieved from <https://ieeexplore.ieee.org/abstract/document/6823132/>
- Hoang, D.-T., & Kang, H.-J. (2019). A survey on deep learning based bearing fault diagnosis. *Neurocomputing*, 335, 327–335.

- Horkoff, J. (2019). Non-functional requirements for machine learning: Challenges and new directions. In *2019 IEEE 27th International Requirements Engineering Conference (re)* (pp. 386–391).
- Huang, X., Ye, Y., Xiong, L., Lau, R. Y., Jiang, N., & Wang, S. (2016). Time series k-means: A new k-means type smooth subspace clustering for time series data. *Information Sciences*, *367*, 1–13.
- Huber, S., Wiemer, H., Schneider, D., & Ihlenfeldt, S. (2019). Dmme: Data mining methodology for engineering applications—a holistic extension to the crisp-dm model. *Procedia Cirp*, *79*, 403–408.
- Idowu, S., Struber, D., & Berger, T. (2021). Asset Management in Machine Learning: A Survey. In *Proceedings - international conference on software engineering* (pp. 51–60). Virtual Event Spain: IEEE. doi: 10.1109/ICSE-SEIP52600.2021.00014
- Ihaka, R., & Gentleman, R. (1996). R: a language for data analysis and graphics. *Journal of computational and graphical statistics*, *5*(3), 299–314.
- Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., & Muller, P.-A. (2019). Deep learning for time series classification: a review. *Data mining and knowledge discovery*, *33*(4), 917–963.
- Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science*, *349*(6245), 255–260.
- Kamishima, T., Akaho, S., & Sakuma, J. (2011). Fairness-aware learning through regularization approach. In *2011 IEEE 11th International Conference on Data Mining Workshops* (pp. 643–650).
- KANG, K., COHEN, S., HESS, J., NOVAK, W., & PETERSON, A. (1990). Feature-oriented domain analysis(foda) feasibility study(final report).
- Kehrer, T., Thüm, T., Schultheiß, A., & Bittner, P. M. (2021). Bridging the gap between clone-and-own and software product lines. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: New ideas and emerging results (icse-nier)* (pp. 21–25). IEEE.
- Kery, M. B., Radensky, M., Arya, M., John, B. E., & Myers, B. A. (2018, April). The Story in the Notebook: Exploratory Data Science using a Literate Programming Tool. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (pp. 1–11). New York, NY, USA: Association for Computing Machinery. Retrieved 2023-02-01, from <https://doi.org/10.1145/3173574.3173748> doi: 10.1145/3173574.3173748
- Kieu, T., Yang, B., & Jensen, C. S. (2018). Outlier detection for multidimensional time series using deep neural networks. In *2018 19th IEEE International Conference on Mobile Data Management (mdm)* (pp. 125–134).
- Kim, M., Zimmermann, T., DeLine, R., & Begel, A. (2017). Data scientists in software teams: State of the art and challenges. *IEEE Transactions on Software Engineering*, *44*(11), 1024–1038.
- Koenzen, A. P., Ernst, N. A., & Storey, M.-A. D. (2020, August). Code Duplication and Reuse in Jupyter Notebooks. In *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (pp. 1–9). Dunedin, New Zealand: IEEE. Retrieved from

<https://doi.org/10.1109/VL/HCC50065.2020.9127202> (ISSN: 1943-6106)
doi: 10.1109/VL/HCC50065.2020.9127202

Kotak, P., & Modi, H. (2020). Enhancing the data mining tool weka. In *2020 5th international conference on computing, communication and security (icccs)* (pp. 1–6).

Kross, S., & Guo, P. J. (2019). Practitioners teaching data science in industry and academia: Expectations, workflows, and challenges. In *Proceedings of the 2019 chi conference on human factors in computing systems* (pp. 1–14).

Krüger, J., & Berger, T. (2020). An empirical analysis of the costs of clone-and platform-oriented software reuse. In *Proceedings of the 28th acm joint meeting on european software engineering conference and symposium on the foundations of software engineering* (pp. 432–444). ACM.

LaToza, T. D., Venolia, G., & DeLine, R. (2006). Maintaining mental models: a study of developer work habits. In *Proceedings of the 28th international conference on software engineering* (pp. 492–501).

Lee, D. J.-L., & Macke, S. (2020). A human-in-the-loop perspective on automl: Milestones and the road ahead. *IEEE Data Engineering Bulletin*.

Lee, K., Kang, K. C., & Lee, J. (2002). Concepts and guidelines of feature modeling for product line software engineering. In *International conference on software reuse* (pp. 62–77).

Leitner, A., Weiß, R., & Kreiner, C. (2012). Analyzing the complexity of domain model representations. In *2012 ieee 19th international conference and workshops on engineering of computer-based systems* (p. 242-248). doi: 10.1109/ECBS.2012.15

Liao, T., Taori, R., Raji, I. D., & Schmidt, L. (2021). Are we learning yet? a meta review of evaluation failures across machine learning. In *Thirty-fifth conference on neural information processing systems datasets and benchmarks track (round 2)*.

Lopez-Herrejon, R. E., & Batory, D. (2001). A standard problem for evaluating product-line methodologies. In *International symposium on generative and component-based software engineering* (pp. 10–24).

Lorenzi, M., Ayache, N., Frisoni, G., Pennec, X., et al. (2010). 4d registration of serial brain's mr images: a robust measure of changes applied to alzheimer's disease. In *Spatio temporal image analysis workshop (stia), miccai* (Vol. 1).

Luo, G. (2016). A review of automatic selection methods for machine learning algorithms and hyper-parameter values. *Network Modeling Analysis in Health Informatics and Bioinformatics*, 5(1), 1–16.

Macías-García, L., Luna-Romera, J. M., García-Gutiérrez, J., Martínez-Ballesteros, M., Riquelme-Santos, J. C., & González-Cámpora, R. (2017). A study of the suitability of autoencoders for preprocessing data in breast cancer experimentation. *Journal of biomedical informatics*, 72, 33–44.

- Mann, S., & Rock, G. (2011). Control variant-rich models by variability measures. In *Proceedings of the 5th international workshop on variability modeling of software-intensive systems* (p. 29–38). New York, NY, USA: Association for Computing Machinery. Retrieved from <https://doi.org/10.1145/1944892.1944896> doi: 10.1145/1944892.1944896
- Mao, Y., Wang, D., Muller, M., Varshney, K. R., Baldini, I., Dugan, C., & Mojsilović, A. (2019). How data scientists work together with domain experts in scientific collaborations: To find the right answer or to ask the right question? *Proceedings of the ACM on Human-Computer Interaction*, 3(GROUP), 1–23.
- Matthew, S., Kyle, K., & Michelle, U. (2018). *Part 2: Scheduling Notebooks at Netflix*. Retrieved from <https://netflixtechblog.com/scheduling-notebooks-348e6c14cfd6>
- Maâzoun, J., Bouassida, N., & Ben-Abdallah, H. (2016). Change impact analysis for software product lines. *Journal of King Saud University - Computer and Information Sciences*, 28(4), 364–380. Retrieved from <https://www.sciencedirect.com/science/article/pii/S1319157816300167> doi: <https://doi.org/10.1016/j.jksuci.2016.01.005>
- McDougal, R. A., Morse, T. M., Carnevale, T., Marengo, L., Wang, R., Migliore, M., ... Hines, M. L. (2017, feb). Twenty years of ModelDB and beyond: building essential modeling tools for the future of neuroscience. *Journal of Computational Neuroscience*, 42(1), 1–10. doi: 10.1007/S10827-016-0623-7
- McNicholas, P. D., Murphy, T. B., & O'Regan, M. (2008). Standardising the lift of an association rule. *Computational Statistics & Data Analysis*, 52(10), 4712–4721.
- McPhillips, T., Bowers, S., Zinn, D., & Ludäscher, B. (2009). Scientific workflow design for mere mortals. *Future Generation Computer Systems*, 25(5), 541–551.
- Mefteh, M., Bouassida, N., & Ben-Abdallah, H. (2015). Implementation and evaluation of an approach for extracting feature models from documented uml use case diagrams. In *Proceedings of the 30th annual acm symposium on applied computing* (p. 1602–1609). New York, NY, USA: Association for Computing Machinery. Retrieved from <https://doi.org/10.1145/2695664.2695907> doi: 10.1145/2695664.2695907
- Merkelbach, S., Von Enzberg, S., Kühn, A., & Dumitrescu, R. (2022). Towards a process model to enable domain experts to become citizen data scientists for industrial applications. In *2022 IEEE 5th international conference on industrial cyber-physical systems (icps)* (pp. 1–6).
- Mohassel, P., & Zhang, Y. (2017). Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)* (pp. 19–38).
- Muller, M., Lange, I., Wang, D., Piorkowski, D., Tsay, J., Liao, Q. V., ... Erickson, T. (2019). How data science workers work with data: Discovery, capture, curation, design, creation. In *Proceedings of the 2019 CHI conference on human factors in computing systems* (pp. 1–15).
- Niehaus, E., Pohl, K., & Böckle, G. (2005). *Software product line engineering: Foundations, principles and techniques, kapitel product management*. Springer, Berlin.

- Nieke, M., Sampaio, G., Thüm, T., Seidl, C., Teixeira, L., & Schaefer, I. (2022). Guiding the evolution of product-line configurations. *Software and Systems Modeling*, 1–23.
- Nieke, M., Seidl, C., & Schuster, S. (2016). Guaranteeing configuration validity in evolving software product lines. In *Proceedings of the tenth international workshop on variability modelling of software-intensive systems* (pp. 73–80).
- Ntoutsis, E., Fafalios, P., Gadiraju, U., Iosifidis, V., Nejdil, W., Vidal, M.-E., . . . others (2020). Bias in data-driven artificial intelligence systems—an introductory survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 10(3), e1356.
- Ordenes, F. V., & Silipo, R. (2021). Machine learning for marketing on the knime hub: The development of a live repository for marketing applications. *Journal of Business Research*, 137, 393–410.
- Passi, S., & Jackson, S. J. (2018). Trust in data science: Collaboration, translation, and accountability in corporate data science projects. *Proceedings of the ACM on Human-Computer Interaction*, 2(CSCW), 1–28.
- Passi, S., & Sengers, P. (2020). Making data science systems work. *Big Data & Society*, 7(2), 2053951720939605.
- Passos, L., Novakovic, M., Xiong, Y., Berger, T., Czarnecki, K., & Wąsowski, A. (2011). A study of non-boolean constraints in variability models of an embedded operating system. In *Proceedings of the 15th international software product line conference, volume 2* (pp. 1–8).
- Patro, S. G. K., & Sahu, K. K. (2015). Normalization: A preprocessing stage. *CoRR*, abs/1503.06462, 1–3. Retrieved from <http://arxiv.org/abs/1503.06462>
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . others (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12, 2825–2830.
- Pereira, J. A., Acher, M., Martin, H., Jézéquel, J.-M., Botterweck, G., & Ventresque, A. (2021). Learning software configuration spaces: A systematic literature review. *Journal of Systems and Software*, 182, 111044.
- Perkel, J. M. (2018, October). Why Jupyter is data scientists’ computational notebook of choice. *Nature*, 563(7729), 145–146. Retrieved 2023-01-26, from <https://doi.org/10.1038/d41586-018-07196-1> doi: 10.1038/d41586-018-07196-1
- Piorkowski, D., Park, S., Wang, A. Y., Wang, D., Muller, M., & Portnoy, F. (2021). How ai developers overcome communication challenges in a multidisciplinary team: A case study. *Proceedings of the ACM on Human-Computer Interaction*, 5(CSCW1), 1–25.
- Pohl, K., Böckle, G., & van Der Linden, F. J. (2005). *Software product line engineering: Foundations, principles and techniques*. Springer Science & Business Media.
- Powers, D. M. (2020). Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061*.

- Rosenmüller, M., & Siegmund, N. (2010). Automating the configuration of multi software product lines. *VaMoS*, *10*, 123–130.
- Rubin, J., Czarnecki, K., & Chechik, M. (2013). Managing cloned variants: a framework and experience. In *Proceedings of the 17th international software product line conference* (pp. 101–110).
- Rule, A., Tabard, A., & Hollan, J. D. (2018). Exploration and explanation in computational notebooks. In *Proceedings of the 2018 chi conference on human factors in computing systems* (p. 1–12). New York, NY, USA: Association for Computing Machinery. Retrieved from <https://doi.org/10.1145/3173574.3173606> doi: 10.1145/3173574.3173606
- Sánchez, A. B., Segura, S., & Ruiz-Cortés, A. (2014). A comparison of test case prioritization criteria for software product lines. In *2014 ieee seventh international conference on software testing, verification and validation* (pp. 41–50).
- Schmidt, D. C. (2006). Guest editor’s introduction: Model-driven engineering. *Computer*, *39*(02), 25–31.
- Schröer, C., Kruse, F., & Gómez, J. M. (2021). A systematic literature review on applying crisp-dm process model. *Procedia Computer Science*, *181*, 526–534.
- Schubert, E., Sander, J., Ester, M., Kriegel, H. P., & Xu, X. (2017). Dbscan revisited, revisited: why and how you should (still) use dbscan. *ACM Transactions on Database Systems (TODS)*, *42*(3), 1–21.
- Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., ... Young, M. (2015). Machine Learning: The High Interest Credit Card of Technical Debt. In G. Zoubin, M. Welling, C. Cortes, N. D. Lawrence, & K. Q. Weinberger (Eds.), *Proceedings of the 28th international conference on neural information processing systems - volume 2* (pp. 2503–2511). Montreal, Canada: MIT Press. Retrieved from <https://ai.google/research/pubs/pub43146>
- Shrestha, N., Botta, C., Barik, T., & Parnin, C. (2020). Here we go again: Why is it difficult for developers to learn another programming language? In *Proceedings of the acm/ieee 42nd international conference on software engineering* (pp. 691–701).
- Siegmund, N., Rosenmüller, M., Kuhlemann, M., Kästner, C., Apel, S., & Saake, G. (2012). Spl conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality Journal*, *20*(3), 487–517.
- Smola, A. (2008). *Introduction to machine learning*.
- Svahnberg, M., & Bosch, J. (1999). Evolution in software product lines: Two cases. *Journal of Software Maintenance: Research and Practice*, *11*(6), 391–422.
- Team, R. C. (2000). R language definition. *Vienna, Austria: R foundation for statistical computing*, *3*(1).

- Tessier, P., Gérard, S., Terrier, F., & Geib, J.-M. (2005). Using variation propagation for model-driven management of a system family. In *International conference on software product lines* (pp. 222–233).
- Thomas, L. S., & Gehrig, J. (2020). Multi-template matching: a versatile tool for object-localization in microscopy images. *BMC bioinformatics*, *21*(1), 1–8.
- Tornhill, A. (2015). *Your Code as a Crime Scene*. Pragmatic Bookshelf. Retrieved from <https://books.google.fr/books?id=17dDnQAACAAJ>
- Van Rijn, J. N., Bischl, B., Torgo, L., Gao, B., Umaashankar, V., Fischer, S., ... Vanschoren, J. (2013). Openml: A collaborative science platform. In *Machine learning and knowledge discovery in databases: European conference, ecml pkdd 2013, prague, czech republic, september 23-27, 2013, proceedings, part iii 13* (pp. 645–649).
- Van Rijn, J. N., & Vanschoren, J. (2015). Sharing rapidminer workflows and experiments with openml. In *Metasel@ pkdd/ecml* (pp. 93–103).
- Vanschoren, J., Rijn, J. N., & Bischl, B. (2015). Taking machine learning research online with openml. In *Workshop on big data, streams and heterogeneous source mining: Algorithms, systems, programming models and applications* (pp. 1–4).
- Vanschoren, J., van Rijn, J. N., Bischl, B., & Torgo, L. (2013). OpenML: Networked Science in Machine Learning. *SIGKDD Explorations*, *15*(2), 49–60. Retrieved from <http://doi.acm.org/10.1145/2641190.2641198> doi: 10.1145/2641190.2641198
- Vanschoren, J., Van Rijn, J. N., Bischl, B., & Torgo, L. (2014). Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, *15*(2), 49–60.
- Vogelsang, A., & Borg, M. (2019). Requirements engineering for machine learning: Perspectives from data scientists. In *2019 IEEE 27th international requirements engineering conference workshops (rew)* (pp. 245–251).
- Vyas, G., & Sharma, A. (2016). Empirical evaluation of metrics to assess software product line feature model usability. *International Journal of Science, Engineering and Computer Technology*, *6*(2), 82.
- Wada, H., Suzuki, J., & Oba, K. (2006). Modeling non-functional aspects in service oriented architecture. In *2006 IEEE international conference on services computing (scc'06)* (pp. 222–229).
- Wang, A. Y., Mittal, A., Brooks, C., & Oney, S. (2019). How data scientists use computational notebooks for real-time collaboration. *Proceedings of the ACM on Human-Computer Interaction*, *3*(CSCW), 1–30.
- Wang, A. Y., Wang, D., Drozdal, J., Liu, X., Park, S., Oney, S., & Brooks, C. (2021, May). What Makes a Well-Documented Notebook? A Case Study of Data Scientists' Documentation Practices in Kaggle. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems* (pp. 1–7). New York, NY, USA: Association for Computing Machinery. Retrieved 2023-01-31, from <https://doi.org/10.1145/3411763.3451617> doi: 10.1145/3411763.3451617

- Wang, D., Liao, Q. V., Zhang, Y., Khurana, U., Samulowitz, H., Park, S., . . . Amini, L. (2021). How much automation does a data scientist want? *arXiv preprint arXiv:2101.03970*.
- Wang, J., Kuo, T.-y., Li, L., & Zeller, A. (2020). Restoring reproducibility of jupyter notebooks. In *Proceedings of the acm/ieee 42nd international conference on software engineering: Companion proceedings* (p. 288–289). New York, NY, USA: Association for Computing Machinery. Retrieved from <https://doi.org/10.1145/3377812.3390803> doi: 10.1145/3377812.3390803
- Williams, G. (2011a). *Boosting*. New York, NY: Springer New York. Retrieved from https://doi.org/10.1007/978-1-4419-9890-3_13 doi: 10.1007/978-1-4419-9890-3_13
- Williams, G. (2011b). Cluster analysis. In *Data mining with rattle and r: The art of excavating data for knowledge discovery* (pp. 179–192). New York, NY: Springer New York. Retrieved from https://doi.org/10.1007/978-1-4419-9890-3_9 doi: 10.1007/978-1-4419-9890-3_9
- Williams, G. (2011c). *Decision trees*. New York, NY: Springer New York. Retrieved from https://doi.org/10.1007/978-1-4419-9890-3_11 doi: 10.1007/978-1-4419-9890-3_11
- Williams, G. (2011d). Model performance evaluation. In *Data mining with rattle and r: The art of excavating data for knowledge discovery* (pp. 307–321). New York, NY: Springer New York. Retrieved from https://doi.org/10.1007/978-1-4419-9890-3_15 doi: 10.1007/978-1-4419-9890-3_15
- Williams, G. (2011e). *Random forests*. New York, NY: Springer New York. Retrieved from https://doi.org/10.1007/978-1-4419-9890-3_12 doi: 10.1007/978-1-4419-9890-3_12
- Williams, G. (2011f). *Support vector machines*. New York, NY: Springer New York. Retrieved from https://doi.org/10.1007/978-1-4419-9890-3_14 doi: 10.1007/978-1-4419-9890-3_14
- Williams, G. (2011g). Transforming data. In *Data mining with rattle and r: The art of excavating data for knowledge discovery* (pp. 149–168). New York, NY: Springer New York. Retrieved from https://doi.org/10.1007/978-1-4419-9890-3_7 doi: 10.1007/978-1-4419-9890-3_7
- Williams, G., et al. (2009). Rattle: a data mining gui for r.
- Wirth, R., & Hipp, J. (2000). Crisp-dm: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining* (Vol. 1, pp. 29–39).
- Xin, D., Ma, L., Song, S., & Parameswaran, A. (2018). How developers iterate on machine learning workflows—a survey of the applied machine learning literature. *arXiv preprint arXiv:1803.10311*.

- Xin, D., Wu, E. Y., Lee, D. J.-L., Salehi, N., & Parameswaran, A. (2021). Whither automl? understanding the role of automation in machine learning workflows. In *Proceedings of the 2021 chi conference on human factors in computing systems* (pp. 1–16).
- Yao, Q., Wang, M., Chen, Y., Dai, W., Li, Y.-F., Tu, W.-W., ... Yu, Y. (2018). Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306*.
- Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., ... Zumar, C. (2018). Accelerating the machine learning lifecycle with MLflow. *IEEE Data Engineering Bulletin*, 41(4), 39–45. Retrieved from <http://sites.computer.org/debull/A18dec/p39.pdf>
- Zainudin, Z., Shamsuddin, S. M., & Hasan, S. (2019). Deep learning for image processing in weka environment. *Int. J. Advance Soft Compu. Appl*, 11(1).
- Zhang, A. X., Muller, M., & Wang, D. (2020). How do data science workers collaborate? roles, workflows, and tools. *Proceedings of the ACM on Human-Computer Interaction*, 4(CSCW1), 1–23.
- Zhang, W., Zhao, H., & Mei, H. (2004). A propositional logic-based method for verification of feature models. In *Formal methods and software engineering: 6th international conference on formal engineering methods, icfem 2004, seattle, wa, usa, november 8-12, 2004. proceedings 6* (pp. 115–130).
- Zöllner, M.-A., & Huber, M. F. (2021). Benchmark and survey of automated machine learning frameworks. *Journal of artificial intelligence research*, 70, 409–472.

Pages web

List of figures

1.1	The relations between multiple ML problems	3
1.2	Acquisition errors	4
1.3	Expert annotations	4
1.4	Drifts in the data	5
2.1	Process diagram showing the relationship between the different phases of CRISP-DM	17
3.1	The rattle classification tab	24
3.2	Rattle clustering tab	24
3.3	Rattle association rules tab	25
3.4	Rattle evaluation tab	25
3.5	The weka decision tree model tab	26
3.6	Weka Knowledge flow layout example	26
3.7	Weka Experimenter tab	27
3.8	Weka explorer tab	27
3.9	Knime workbench tab	29
3.10	Knime Node ports and node status	29
3.11	OpenML home page	31
3.12	Knime hub workflows repository	32
4.1	Mandatory feature group	37
4.2	Optional feature group	38
4.3	Alternative feature group	38
4.4	Or feature group	38
4.5	GPL feature model	39
5.1	washing machine motor engine time series	49
5.2	Scenario 4 automation level 1 workflow steps	54
5.3	Scenario 1 results	56
5.4	Scenario 2 results	56
5.5	Scenario 3 results	57
6.1	Contextualized Solutions within the Overall Solution Space	64
6.2	Process overview	66
6.3	Functional overview of our approach	67
6.4	Tool UI for Business Requirements configuration	68
6.5	Knowledge partitioning into feature model spaces	69
6.6	Company tool overview	74
6.7	Features and constraints	75

7.1	Feature model to illustrate definitions and metrics	84
7.2	Three main steps of our SPL construction	90
8.1	Fate properties form	100
8.2	Fate properties branch	101
8.3	Prompting GPT3 to read configurations and ask questions	101
8.4	Answering the chat questions	102
8.5	Asking the chat further explanations of the options on anomaly types	104
8.6	Summary of provided answers	105

List of definitions

1.5.1	Software product lines	7
1.5.2	Commonality	7
1.5.3	Variability	7
1.5.4	Configuration	7
1.5.5	SPL and Feature Model	7
1.5.6	configuration	7
1.5.7	sub-space	7
1.5.8	partial configuration	7
1.5.9	a domain	7
1.5.10	olution space	7
1.5.11	product	8
1.5.12	application products	8
1.5.13	ML workflow product	8
6.2.1	67
7.2.1	Equivalence Class on A according to a subset of features of FM	83
7.2.2	Problem equivalence classes	83
7.2.3	Solution equivalence classes	83
7.2.4	Unifiable classes	84

List of examples

Listes des algorithmes

Appendices

Faciliter l'inclusion humaine dans le processus de science des données : de la capture des exigences métier à la conception d'un workflow d'apprentissage automatique opérationnel

Yassine EL AMRAOUI

Résumé

Le processus de création de workflows d'apprentissage automatique par les data scientists implique de comprendre les besoins métier, d'analyser les données et d'expérimenter pour trouver des solutions. Cependant, cette approche par essais et erreurs manque de structure pour prendre en compte toutes ces dimensions, et conduit souvent à des essais inadaptés et à des interprétations limitées des résultats par les praticiens. Pour remédier à ce problème dans le cadre de la conception de workflows destinés à la détection d'anomalies dans les séries temporelles, notre travail propose trois contributions principales :

- Prise en compte des données, des exigences métiers et des composants de la solution dans la conception des workflows de machine learning
- Personnalisation des workflows pour des solutions sur mesure en s'appuyant sur des descriptions partielles et modulaires du problème sous forme de configurations.
- Enrichissement de la connaissance portée par les lignes de produits logiciels par l'exploitation des produits. Ces contributions visent à rationaliser la création de workflows d'apprentissage automatique, en promouvant une approche plus structurée, le partage des connaissances et un meilleur alignement sur l'analyse du problème avant de rechercher des applications similaires.

Mots-clés : Ligne de produit logiciel, Science des données, Inclure l'humain dans la boucle

Abstract

The process of creating machine learning workflows by data scientists involves understanding business needs, analyzing data and experimenting to find solutions. However, this trial-and-error approach lacks the structure to take all these dimensions into account, and often leads to inadequate testing and limited interpretations of results by practitioners. To address this problem in the context of designing workflows for anomaly detection in time series, our work proposes three main contributions:

- Consideration of data, business requirements and solution components in the design of machine learning workflows.
- Customization of workflows for tailor-made solutions, based on partial and modular descriptions of the problem in the form of configurations.
- Enriching the knowledge carried by software product lines through product exploitation.

These contributions aim to streamline the creation of machine learning workflows, promoting a more structured approach, knowledge sharing and better alignment with problem analysis before searching for similar applications.

Keywords: Software product line, Data science, Data science, Human in the loop