



Universität Stuttgart

Concepts and Methods for the Design, Configuration and Selection of Machine Learning Solutions in Manufacturing

Von der Graduate School of Excellence advanced Manufacturing
Engineering der Universität Stuttgart zur Erlangung der Würde eines
Doktors der Ingenieurwissenschaften (Dr.-Ing.) genehmigte Abhandlung

Vorgelegt von
Alejandro Gabriel Villanueva Zacarias
aus Puebla, Mexiko

Hauptberichter: Prof. Dr.-Ing. habil. Bernhard Mitschang
Mitberichter: Prof. Dr.-Ing. Norbert Ritter

Tag der mündlichen Prüfung: 21.12.2021

Institut für Parallele und Verteilte Systeme (IPVS)
der Universität Stuttgart

2021

ACKNOWLEDGEMENTS

"Tú que todo lo sabes y lo que no lo inventas..."

(You, who knows it all and what you don't know, you invent...)

Back when I was a kid, my classmates used to precede any question they had for me with the quote above. Throughout the years that sentence has reminded me that the answers to some questions may still need to be invented and that I, perhaps, could be able to do just that if I learn enough about them. This is my way to make justice to my classmates' claims.

This dissertation is the result of my doctoral project at the Graduate School of Excellence advanced Manufacturing Engineering (GSaME), in collaboration with the Application Software department from the Institute of Parallel and Distributed Systems at the University of Stuttgart. The interdisciplinary nature of GSaME offered me the perfect platform to combine my research interest in computer science, statistics, quality control, machine learning and manufacturing and to orient them towards industrial applications. My doctoral project would not have reached its potential without the help, resources and opportunities that GSaME, its professors and staff offered me.

Several people assisted me throughout the years on the path to develop my research skills in order to complete this dissertation. Special thanks to my supervisor *Prof. Dr. Bernhard Mitschang* for his unwavering support right from the start of the doctoral project. His guidance and encouragement to explore, but without forgetting to remain *positive and productive*, proved key to complete this dissertation as well as the project in the context of the *Software Campus* initiative. I would also like to extend my sincere thanks to *Prof. Dr. Norbert Ritter* for the supervision of this dissertation, as well as for his collaboration as examiner during the doctoral defense.

Many thanks to *Dr. Sylvia Rohr, Hans-Friedrich Jacobi, Prof. Alfred Katzenbach, Prof. Dr. Georg Herzwurm* and *Dr. Laura Gern* for their valuable contributions to my development as a researcher. Your input, questions and advice has helped me expand my horizons.

I am also grateful to the many colleagues at the department of Application Software for their daily collaboration and valuable lessons. In particular, I would like to thank *Peter Reimann, Christian Weber, Michael Behringer, Manuel Fritz, Cornelia Kiefer* and *Rachaa Ghabri* for their feedback, help and collaboration at different stages of the research project. Special thanks to *Peter Reimann* and *Christian Weber* for proofreading the first drafts of this dissertation. I also had the pleasure to work with *Holger Schwarz, Pascal Hirmer, Julian Ziegler, Corinna Giebler, Vitali Hirsch, Marco Spieß, Dennis Przytarski, Yannick Wilhelm* and *Jan Königsberger*. Their advice, company and sometimes patience made the daily work at university more interesting and enjoyable.

I also wish to thank my cooperation partner within the Software Campus project, *Ingo Sawilla*. His insights, conversation and feedback regarding leadership and project management were invaluable input not just for the development of the GUACAMOLE prototypes, but also for my future career.

I cannot begin to express my thanks to my family, who has stood and continues to stay by my side offering me love, support and freedom to follow my dreams, wherever they may take me. To my brother for his company throughout the games, years and circumstances. Life would not be the same without you and your talent to bring out the playful kid I still carry around.

Thank you for listening to my ideas in their rawest form. To my father for opening my eyes to life abroad, with its highs and lows. I recognize your influence in me as I face challenges similar to the ones you must have faced back then during your travels, especially when the line between visitor and local starts to blur. I draw inspiration from you whenever I need to recollect myself. To my mother for teaching me to remain curious, courageous and disciplined regardless of whatever life brings. Throughout the years, you taught me many of the research skills I use now. From the reading sessions on your lap, the fruit experiments to understand homework, the leisure visits to the laboratories, libraries and bookshops, to the creative high school projects with complicated executions. It all led to this. While working on this dissertation, I began to finally understand your interest in research. My deepest thanks to you, my first and greatest supervisor.

Stuttgart, November 2021
Alejandro Gabriel Villanueva Zacarias

CONTENTS

1. Introduction	15
1.1. Motivation	16
1.2. Application Context of ML Solution Development in Manufacturing	19
1.2.1. Definition of Machine Learning Solution	19
1.2.2. The Project Scope	20
1.2.3. The Development Process	21
1.2.4. Roles in the Development Team	23
1.2.5. Data and Available Tools	25
1.3. Research Challenges	26
1.3.1. Design Challenge	27
1.3.2. Configuration Challenge	28
1.3.3. Selection Challenge	29
1.3.4. Process Challenge	30
1.4. Research Contributions	30
1.4.1. Research Contribution for the Design Challenge	32
1.4.2. Research Contribution for the Configuration Challenge	32
1.4.3. Research Contribution for the Selection Challenge	33
1.4.4. Research Contribution for the Process Challenge	34

1.5. Dissertation Outline	35
2. Theoretical Background	37
2.1. Characteristics of Production System Design	37
2.1.1. Production System Design within the Product Conception Phase	38
2.1.2. Classification Systems	39
2.1.3. Feature Technology	40
2.1.4. Model-based Systems Engineering	41
2.2. Data in Manufacturing Use Cases	42
2.3. Machine Learning	45
2.4. Approaches to Design ML Solutions	47
2.5. Approaches to Configure ML Solutions	48
2.6. Approaches to Select ML Solutions	49
3. ML Solution Framework	51
3.1. Related Work	53
3.1.1. Challenges of ML Solutions in Manufacturing	53
3.1.2. Empirical Evaluation of Machine Learning Algorithms ..	54
3.1.3. Automated Machine Learning	54
3.2. Design Requirements	55
3.2.1. Specification of a Domain-specific Problem Definition ..	55
3.2.2. Efficient and Accurate Data Utilization	56
3.2.3. Consideration of Available IT Resources	56
3.2.4. Property-based and Performance-based Selection and Configuration of ML Algorithms	57
3.2.5. Enhanced ML Solution Comprehensibility	58
3.3. ML Solution Framework	58
3.3.1. ML Solution Development Process	59
3.3.2. Metadata Profiles	63
3.3.3. ML Solution Viewer	66
3.4. Prototypical Implementation and Discussion	67
3.5. Summary and Future Work	71

4. Axiomatic Design for Machine Learning	73
4.1. Problem Context and Requirements	75
4.1.1. Use Case: Fault Detection in a Production Line	75
4.1.2. Feasibility Requirements in the Design of ML Solutions	76
4.2. Main Concepts of Axiomatic Design.....	78
4.3. Axiomatic Design for Machine Learning (AD4ML)	82
4.3.1. Adaptations to Axiomatic Design for ML Solutions.....	82
4.3.2. AD4ML Specification for the Fault Detection Use Case ..	85
4.3.3. Visualization of ML Solution Specifications	91
4.3.4. Reusability of Specification Components	94
4.3.5. Agile Design of ML Solutions	95
4.4. Approaches to Validate and to Assess ML solution Specifications During the Design Process.....	96
4.4.1. Validation of ML Solution Specifications.....	96
4.4.2. Assessment of ML Solution Specifications	99
4.5. Prototypical Implementation	101
4.6. Discussion and Assessment	105
4.6.1. R_1 . Enablement of Systematic Experimental Learning ..	105
4.6.2. R_2 . Clear and Objective Documentation of the Design Intention	106
4.6.3. R_3 . Support of End-to-end Traceability and Consistency	107
4.7. Related Work	107
4.8. Summary and Future Work.....	111
5. AssistML.....	113
5.1. Application scenario for ML Solution recommendations	115
5.1.1. Reusing ML Solutions for Predictive Use Cases	115
5.1.2. Practical Requirements	116
5.2. Related Work	118
5.2.1. AutoML Systems	118
5.2.2. Meta-Learning	119
5.2.3. Explainable AI	121
5.3. AssistML Metadata Repository.....	122

5.4. AssistML: A Concept to Recommend ML Solutions	126
5.4.1. Step 1: Select ML Solutions on Data Similarity	127
5.4.2. Step 2: Identify Acceptable/Nearly Acceptable ML Solutions	130
5.4.3. Step 3: Find ML Solution Patterns	134
5.4.4. Step 4: Generate List of Recommendations	136
5.5. Prototype and Evaluation	137
5.5.1. Prototypical Implementation	137
5.5.2. Evaluation Approach	140
5.5.3. Evaluation Results	143
5.5.4. Assessment	153
5.6. Summary and Future Work	154
6. Conclusion and Future Work	157
6.1. Assessment of the Research Contributions	157
6.2. Future Research Directions	162
6.2.1. ML Solution Assessment Function	162
List of Author Publications	163
Bibliography	169
List of Figures	179
List of Tables	185
 Appendices	 188
A. Sample metadata profiles	189
B. System Requirements and Installation Guide for AssistML	191

ZUSAMMENFASSUNG

Die Anwendung von Verfahren des maschinellen Lernens (ML) findet in Produktionsunternehmen immer häufiger statt. Entwicklungsteams werden beauftragt, ML-Lösungen zur Unterstützung individueller Anwendungsfälle umzusetzen. Mit dem Begriff **ML-Lösung** bezeichnen wir eine Menge an Softwarekomponenten und Lernalgorithmen, die eine prädiktive Fähigkeit bieten. ML-Lösungen basieren auf Anwendungsfalldaten, (Hyper) Parametern und technischen Konfigurationen.

Aktuell sind Entwicklungsteams mit vier Herausforderungen konfrontiert, welche den Entwicklungsprozess für ML-Lösungen erschweren. Erstens, es mangelt an einem formellen Ansatz zur Spezifikation von ML-Lösungen, der die Auswirkung einzelner Lösungskomponenten auf domänenspezifische Anforderungen nachweist. Zweitens, es mangelt an einem Ansatz zur Dokumentation von ML-Lösungskonfigurationen, mit welchen die erzielten Ergebnisse reproduzierbar werden. Drittens, es mangelt an einem Ansatz zur Empfehlung und Auswahl von ML-Lösungen, der für nicht ML-Experten intuitiv ist. Viertens, es mangelt an einer ausführlichen Schrittreihenfolge, die den Einsatz von Best Practices sowie die Betrachtung technischer und domänenspezifischer Aspekte während des Entwicklungsprozesses gewährleistet. Die Nichtbeachtung obiger Herausforderungen führt insgesamt zu

längeren Entwicklungszeiten und höheren Kosten, sowie zu ungeeigneten ML-Lösungen, welche schwer zu verstehen und wiederzuverwenden sind.

Diese Dissertation stellt Konzepte vor, um diese Herausforderungen zu bewältigen. Dies sind das axiomatische Design für maschinelles Lernen (AD4ML), das Profiling-Rahmenwerk für ML-Lösungen und AssistML.

Axiomatisches Design für maschinelles Lernen (AD4ML) ist ein Konzept zum strukturierten und agilen Entwurf von ML-Lösungsspezifikationen. AD4ML sorgt für einen eindeutigen Bezug zwischen domänenspezifischen Anforderungen und konkreten Softwarekomponenten. AD4ML-Spezifikationen können somit vor der Implementierung hinsichtlich der Anforderungen von Domänenexperten validiert werden.

Das Profiling-Rahmenwerk für ML-Lösungen dokumentiert Metadaten von ML-Lösungen zur Beschreibung der Datenmerkmale, technischen Konfigurationen und Parameterwerte von Softwarekomponenten sowie mehrere Leistungsmetriken. Die Metadaten stellen die Basis für die Reproduzierbarkeit entwickelter ML-Lösungen dar.

Das Konzept AssistML empfiehlt ML-Lösungen für neue Anwendungsfälle. AssistML sucht aus dokumentierten ML-Lösungen diejenigen, die die Leistungsanforderungen des neuen Anwendungsfalls am besten erfüllen. Die ausgewählten Lösungen werden dann den Entscheidungsträgern einfach und intuitiv vorgestellt.

Jedes dieser Konzepte wurde ausgewertet und prototypisch umgesetzt. Zusammen bieten diese Konzepte einen technologieunabhängigen Ansatz zur Entwicklung von ML-Lösungen. Dadurch können ML-Lösungskomponenten schnell identifiziert und wiederverwendet werden. Aussagekräftige Erklärungen für Experten und Nicht-Experten werden ebenso bereitgestellt. Diese Vorteile führen zu kürzeren Entwicklungszeiten, geringerem Ressourceneinsatz pro Entwicklungsprojekt und fundierten Entscheidungen zur Entwicklung und Auswahl von ML-Lösungen.

ABSTRACT

The application of Machine Learning (ML) techniques and methods is common practice in manufacturing companies. They assign teams to the development of ML solutions to support individual use cases. This dissertation refers as **ML solution** to the set of software components and learning algorithms to deliver a predictive capability based on available use case data, their (hyper) parameters and technical settings.

Currently, development teams face four challenges that complicate the development of ML solutions. First, they lack a formal approach to specify ML solutions that can trace the impact of individual solution components on domain-specific requirements. Second, they lack an approach to document the configurations chosen to build an ML solution, therefore ensuring the reproducibility of the performance obtained. Third, they lack an approach to recommend and select ML solutions that is intuitive for non ML experts. Fourth, they lack a comprehensive sequence of steps that ensures both best practices and the consideration of technical and domain-specific aspects during the development process. Overall, the inability to address these challenges leads to longer development times and higher development costs, as well as less suitable ML solutions that are more difficult to understand and to reuse.

This dissertation presents concepts to address these challenges. They are Axiomatic Design for Machine Learning (AD4ML), the ML solution profiling framework and AssistML.

AD4ML is a concept for the structured and agile specification of ML solutions. AD4ML establishes clear relationships between domain-specific requirements and concrete software components. AD4ML specifications can thus be validated regarding domain expert requirements before implementation.

The ML solution profiling framework employs metadata to document important characteristics of data, technical configurations, and parameter values of software components as well as multiple performance metrics. These metadata constitute the foundations for the reproducibility of ML solutions.

AssistML recommends ML solutions for new use cases. AssistML searches among documented ML solutions those that better fulfill the performance preferences of the new use case. The selected solutions are then presented to decision-makers in an intuitive way.

Each of these concepts was evaluated and implemented. Combined, these concepts offer development teams a technology-agnostic approach to build ML solutions. The use of these concepts brings multiple benefits, i. e., shorter development times, more efficient development projects, and better-informed decisions about the development and selection of ML solutions.

INTRODUCTION

The increasing complexity of products and production processes demands the use of new technologies to continue manufacturing high-quality products with financially efficient production costs (Westkämper and Löffler, 2016). Machine Learning (ML) is one such technology. Manufacturing companies apply ML to handle the complexity of their production systems (Wuest et al., 2016). ML is not applied in isolation. It has to be deployed within bigger software systems, which are in turn embedded in the production system (Kuwejima et al., 2020; Sculley et al., 2015). This complex deployment environment, i. e., in a *system of systems*, along with other factors make the current development of software systems with ML software error-prone and time-consuming. As a consequence, it is difficult to derive benefits from ML deployment in the production system and to speed up ML adoption to meet the demand of manufacturing companies.

This dissertation identifies and addresses four main challenges that come up during the development of ML solutions. In response to these challenges, it provides concepts and methods to facilitate the design, configuration and selection of ML solutions in manufacturing use cases.

The remainder of this chapter is organized as follows: [Section 1.1](#) introduces the current state of developing systems with ML for manufacturing cases that motivate this dissertation. [Section 1.2](#) introduces the application context that all contributions consider, particularly its characteristic components. It also introduces the concept of ML solutions. [Section 1.3](#) introduces the four research challenges that this dissertation addresses. [Section 1.4](#) summarizes the four contributions of this dissertation to address the research challenges. [Section 1.5](#) describes the outline of the remaining dissertation.

1.1. Motivation

The application of ML is common in manufacturing companies (Choudhary et al., 2009; Pham and Afify, 2005). It is also known to be rapidly evolving over the years. Several factors, technological and economical, have led to a wide diversification in the approaches to apply ML in manufacturing. This dissertation focuses on the development aspects of applying ML in manufacturing. Thus, this section describes the factors that characterize ML development projects in manufacturing companies.

Increased complexity of production systems. Companies set up more complex production systems to produce more complex products as demanded (Westkämper and Löffler, 2016). The complexity in the production system can be reflected in e. g., more production steps, additional equipment required, or higher quality requirements (Moyné and Iskandar, 2017). This additional complexity demands a higher skill level from workers to perform their tasks correctly. Besides training, which is time-consuming and expensive, systems with ML can also reduce the complexity for workers in multiple ways.

This technological factor however increases the complexity of ML development projects. The complexity of the production systems also complicates the integration of ML into these production systems.

Increased availability of manufacturing data. Data is produced in vast quantities at many steps in a production system (Moyne and Iskandar, 2017; Pham and Afify, 2005) and is stored in multiple formats (Gupta, 2018; O'Donovan et al., 2015). Examples include product catalogs, production scheduling, production orders, part and material stocks. They are stored in relational databases, data lakes, data warehouses or (distributed) file systems (O'Donovan et al., 2015). These data sources can contain relational data, images, video, time series or unstructured text (Mäkinen et al., 2021). This diversity in manufacturing data can be characterized with the five V's of big data (Gandomi and Haider, 2014; Moyne and Iskandar, 2017; Tao et al., 2018). The acronym stands for five characteristics of so-called *big* data sets, i. e., volume, variety, velocity, veracity and value. Each characteristic imposes challenges to the use of data to inform decision-making (see [Section 2.2](#)).

Overall, the increased availability of data has a mixed impact on ML development projects. On the one hand, comprehensive data collection about customers, employees, suppliers, products, equipment, and processes facilitates the training of comprehensive ML models. On the other hand, the heterogeneous data structures, proprietary systems and closed data formats (O'Donovan et al., 2015) complicate the implementation of data preprocessing pipelines and data analysis. This is because general purpose data analysis software may not be ready (O'Donovan et al., 2015) to integrate heterogeneous proprietary data sources. This leads to additional development efforts and overall to more complex software systems.

Increased availability of ML software and tools. ML software and tools are released constantly (Atwal, 2020). They cover a wide spectrum of software, from programming or scripting languages, execution platforms, middleware, and ML libraries. Also, some of this new software specializes in a particular task, e. g., data collection, data processing, model training, or deployment. This means that ML development projects need to combine many tools to produce a working software system that spans all these different tasks. In that situation, some components handle, e. g., data preparation,

while others train an ML model and others trigger communications with other systems based on the ML model's predictions. Additionally, integration code is required to make all these components work together.

Therefore, the specialization of software components leads to more complex software systems, i. e., a so-called technical debt (Sculley et al., 2015). This is manifested in additional efforts to implement and maintain the software system due to the many compatibility checks needed. It also makes it difficult to unify the technological choices of different ML development projects, as development teams can use components that only fulfill their own specific use case needs.

Scarcity of ML experts. The ever-growing number and complexity of available tools and the new algorithms that research develops makes it hard to train enough experts to apply ML tools (Zaharia et al., 2018). This factor leads to a scarcity of ML experts (Flaounas, 2017), i. e., data scientists and data engineers. This slows down the execution of development projects or the depth of the analysis carried out for each individual project. Companies have created new roles to cope with this lack of experts. Citizen data scientists are domain experts with some knowledge of data science and machine learning (Gröger, 2018). Workers in this role are first and foremost domain experts, but also complement the work of data scientists. They can apply existing ML solutions and assess the utility of ML solutions for the use case, but cannot develop complex ML solutions independently.

The unequal level of knowledge and skills between ML experts and citizen data scientists presents a problem for the standardization of software components across multiple development projects. Citizen data scientists may give preference to custom ML solutions that are easily applicable to their use case over more complex and generic ones (Xin et al., 2021). As a result, the scarcity of ML experts has a negative impact on ML development projects.

1.2. Application Context of ML Solution Development in Manufacturing

This section presents an exemplary application context in which manufacturing companies develop ML solutions. The first part defines the concept of an ML solution to detail the output that manufacturing companies expect from ML development projects. Subsequent parts of this section elaborate on other application context elements needed to build an ML solution. This includes the project scope, the development process, the team roles, the available resources and systems. Thereby, this application context becomes a source of valid assumptions to define concrete research challenges. Information about all elements in the application context are based on relevant literature to ensure its correctness and applicability.

1.2.1. Definition of Machine Learning Solution

Development projects are launched to build a software system that can deliver predictive insights for a specific use case (Braschler et al., 2019). Such a system must be able to read the available data and produce predictions that can help the use case, i. e., any of the supervised machine learning outputs, e. g., classification or regression (for more, see [Section 2.3](#)). This dissertation proposes the concept of ML solution to refer to the complete end-to-end software system that is built in the development project. Specifically, an ML solution is a coherent sequence of deployed software components, ML algorithms, their configurations, and domain-specific specifications that are necessary to deliver a predictive capability at a use case based on available data. [Figure 1.1](#) exemplifies this conceptual definition of an ML solution. The most commonly known *core components* of the ML solution are shown with double line borders. These include (1) components for data collection, (2) data feature encoding and selection, (3) a trained ML model, and (4) data post-processing or reporting. All these components have custom values specific for their intended use case. In addition to them, other components are still required for the ML solution to be functional. These (5) *extended*

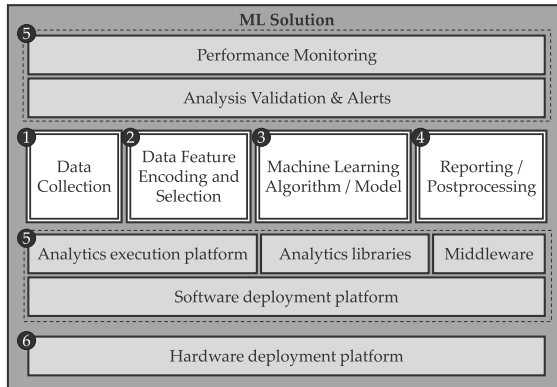


Figure 1.1.: Elements of an ML solution. Core components have double line borders.

components (shown in light gray and with single line borders) include software dependencies, e. g., specific libraries and execution platforms from which generic ML algorithms are used. The ML solution also needs to integrate with other existing systems that are available at the use case. This requires communication components, e. g., middleware. Moreover, the ML solution can be expected to perform within the constraints of the overall production system. To verify this, monitoring or reporting components need to be included as well. Underlying all software components is (6) the hardware infrastructure platform on which the ML solution is deployed. Overall, this example ML solution illustrates that solutions include many more software components in addition to a trained ML model (Sculley et al., 2015).

1.2.2. The Project Scope

ML solution development projects can have different scopes. Multiple authors (Monostori, 2003; Sharp et al., 2018) agree that most projects apply ML on individual production steps. Sharp et al., 2018 takes the hierarchy proposed in the the ISA-95 framework to illustrate the granularity of ML applications. The ISA-95 framework covers five levels from the low-level

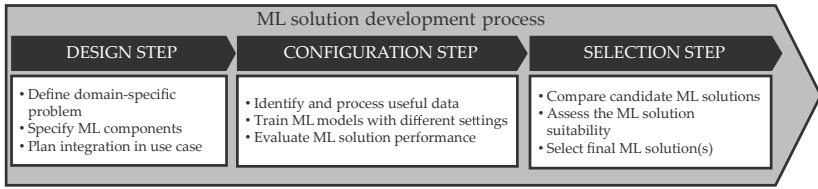


Figure 1.2.: Typical ML solution development process.

manufacturing problems (level 0) to the most abstract, high-level ones (level 4). Sharp et al., 2018 and Monostori, 2003 agree that ML applications focus on single production steps, e. g., predicting machine failures to schedule maintenance or automatically detecting product defects. Bigger scopes, e. g., production scheduling or operations management, although possible, have not seen widespread adoption. Therefore, the application context in this dissertation focuses on use cases for single production steps. Concretely, the application context focuses on predictive use cases that can be addressed with supervised learning.

1.2.3. The Development Process

The application context distinguishes three main steps in the ML solution development process, i. e., design, configuration and selection steps. This division combines the viewpoints from academia (Ashmore et al., 2021; Mikkonen et al., 2021; Pham and Afify, 2005; Weber et al., 2019) and industry (Bernardi et al., 2019; Flaounas, 2017) about ML development projects in industry. The three steps group activities that all projects perform when building an ML solution, regardless of the development methodology they follow. Figure 1.2 shows the three steps in sequence along with brief descriptions of their main activities.

The design step concerns the definition of a domain-specific problem that guides ML solution development. This first step determines which problem from the production system can be solved with ML. It sets the scope for the whole project (Bernardi et al., 2019). The domain-specific problem becomes

a guideline to assess which ML solution components can be useful, and how they should be integrated in the use case. Depending on the approach, the problem definition may be done more or less formally. For instance, CRISP-DM requires the problem to be stated as a series of business goals and plans (Shearer, 2000). Meanwhile, practitioners may simply assess the cost-benefit ratio of implementing an ML solution given the resources available for the use case (Flaounas, 2017).

The configuration step refers to the technical development of the ML solutions. This step is characterized by the iterative training of several ML models using different settings. For this purpose, available data sources are explored and analyzed to obtain a set of useful data features. These data features are then used together with an ML algorithm to train a predictive model, e. g., a classifier or regressor. This step requires multiple attempts and adaptations to make the available data and ML components work together (Flaounas, 2017). In this process, different settings, i. e., seed values, hyper-parameter values, or sampling techniques are used. As a consequence, multiple candidate ML solutions are developed for the same use case, each one with small differences in the data features, the software libraries, the learning algorithms or the settings it uses. Only ML solutions with performance suitable for the use case can be considered candidates. Therefore, the performance evaluation is fundamental to complete the model training (Ashmore et al., 2021; Bernardi et al., 2019; Weber et al., 2019).

The selection step refers to the assessment of multiple candidates and the selection of an ML solution. During the configuration step, the performance of each ML solution is tested against the use case expectations. In the selection step, all candidate ML solutions are compared with one another to determine which one is more suitable for the use case. Use case suitability is a complex question that cannot be measured with a single performance metric (Paley et al., 2020). Additional aspects like scalability, explainability, reliability or even a combination of many performance metrics can be equally decisive. This selection step is typically addressed indirectly during deployment (Ashmore et al., 2021; Bernardi et al., 2019; Flaounas, 2017). In the end, only the most suitable ML solution(s) are selected for use.

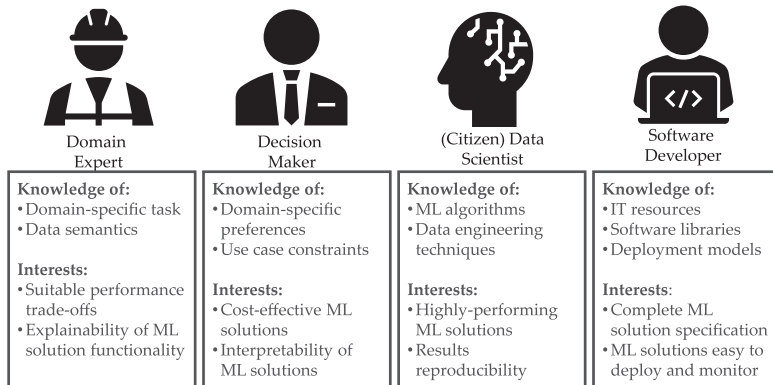


Figure 1.3.: Typical roles in ML solution development team.

1.2.4. Roles in the Development Team

ML solution development projects are executed by a multi-disciplinary team. Each team member plays a role during the development process according to his/her specialized knowledge and interests. Figure 1.3 shows the four roles present in the application context. Please notice that each role can be assigned to multiple people depending on the size of the project.

Domain expert. This role is considered the direct user of the ML solution. It is assumed by people who operate the use case activities. This applies to any person involved in productive processes throughout the product lifecycle. Thus, production engineers, assembly line workers, or quality assurance specialists typically assume this role. Domain experts are involved in the development project of an ML solution during the design and selection steps. They help define the task that the ML solution must fulfill to support the use case. They also are the ultimate source regarding the interpretation of use case data and of its semantics. Domain experts desire an ML solution that delivers explainable results with overall good performance.

Decision maker. This role leads and takes responsibility over the ML solution development project. Therefore, the role can only be assumed by people with decision power in the organization, e. g., product owners,

production managers, or c-suite executives. They are involved in the selection step of the development process. Their decision power determines the overall project constraints, i. e., the available financial budget and time frame for the development. Decision makers also set the performance preferences that the resulting ML solution should fulfill. These preferences correspond to the domain-specific goals that the use case needs to meet to be considered productive. For instance, the decision maker may decide to accept a running time of 5 seconds per prediction in order to keep the use case under control in takt time. Finally, decision makers desire ML solutions that can support the use case in a cost-effective manner. Also important for them is the ability to explain the functioning of the ML solution, particularly when its results contradict the judgment of domain experts.

Data scientist. This role is the main designer of the ML solution. The role is assumed by people with knowledge of machine learning and its related disciplines, like data engineering and statistics. They carry out most activities in the configuration step and collaborate with other roles in the other two steps. They contribute their knowledge to design the ML solution, either to implement it themselves afterwards or to provide a specification for the software developers to follow. Therefore, their main interest is to find highly-performing ML solutions, e. g., ML solutions that deliver as many correct predictions as possible in the shortest possible time. A second interest for data scientists is the reproducibility of the results they obtain. The ML solutions they design should perform in production in the same way as they did during development. For this reason, data scientists are interested in making sure that the designs they propose are reliable and robust.

Software developer. This role is mainly responsible for building and deploying the software system of an ML solution, i. e., implementing ML solution designs. This role is assumed by software engineers, programmers or IT specialists, i. e., people with knowledge of the company's hardware and software resources. The software developer is involved in the configuration step of the development process in collaboration with the data scientist. Besides implementing ML solutions, the software developer also verifies the viability of the designs proposed by the data scientists. He or she verifies

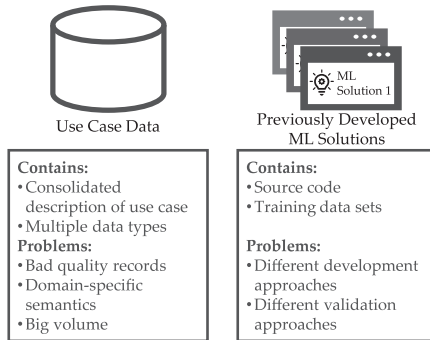


Figure 1.4.: Common elements in ML solution development.

that ML solutions can be deployed in the hardware and software available to the use case. In order to do that, he or she is interested in a complete specification of all hardware, software and data dependencies of the ML solution. Therefore, he or she is interested in developing ML solutions that are easy to deploy and monitor.

1.2.5. Data and Available Tools

Every development project must secure a data source to develop ML solutions. The application context assumes that use case data consists of records from multiple sources consolidated into a single one. This excludes any data discovery that takes place before beginning the development process. Yet, the consolidated data set can contain different data types, e. g., qualitative and quantitative data from relational databases, unstructured text from document archives or time series data from sensors. These data features may also present quality problems, such as missing values, badly-formatted text, outliers, or invalid values. Moreover, the application context assumes that data features may have semantics that cannot be directly captured in the data source. For instance, encoding dates in UNIX format gives them the appearance of numerical data. Also, the use case data can be difficult to

manage after a certain dimension, e. g., if historical data from several years is collected.

Finally, the application context assumes that companies carry out multiple developments (Bernardi et al., 2019), as they seek to support many use cases with ML solutions. The application context thus assumes that finished ML solutions, i. e., its documentation and components, are stored and available. This includes the source code or software components of previous ML solutions, some description about them and samples of the data used to train its ML model. Beyond these contents, the application context does not assume any standardization in the way the solutions were developed or validated. This assumption reflects the limited scope of development projects, which only consider one use case at the time.

1.3. Research Challenges

The application context described in the previous section characterizes typical situations and approaches to ML solution development in manufacturing. This dissertation identifies four challenges in this application context that render ML solution development ineffective. Ineffective ML solution development projects are characterized by longer development times, duplicate tasks, duplicate software implementations and/or duplicate data sources, unclear team collaboration, and difficulties to understand, compare and integrate ML solutions or their components to their target use case. The lack of methods and concepts to address the four challenges represents a problem for ML solution development. This dissertation addresses the problem of ineffective ML solution development by proposing concepts and methods to conquer the design, configuration, selection and process challenges.

This section formulates the four unaddressed research challenges that constitute the research problems in focus here. [Figure 1.5](#) shows the scope of all challenges in the context of the ML solution development process.

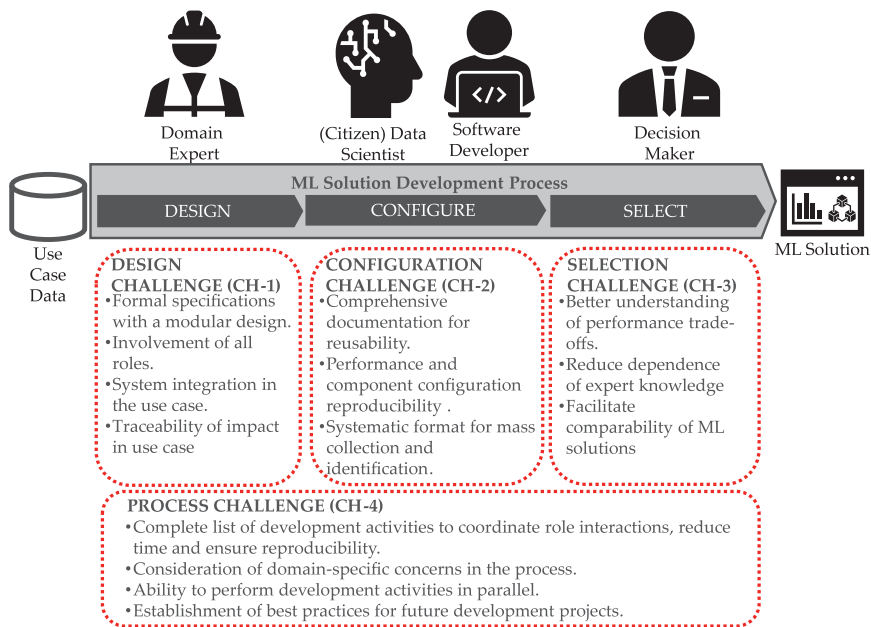


Figure 1.5.: Detail of the research challenges occurring during the development process of ML solutions.

1.3.1. Design Challenge

The first challenge pertains to the design step in the ML solution development process. Typically, the data scientist is in charge of the problem definition for the new ML solution. Existing specification methods (see [Section 2.4](#)) make it difficult to distinguish between the technical specification of components and the domain-specific needs that motivate the use of ML (Viaene and Van den Bunder, 2011). This is partly due to the fact that the different roles of the development team, domain experts in particular, lack a clear way to collaborate with data scientists to provide useful input for the ML solution specification. Moreover, specification documents tend to be unstructured in nature. This makes it difficult to understand the functioning of its software components, i. e., the need of certain software components to solve the

domain-specific problem. It also makes it hard to compare and reuse existing specifications. The lack of structure also removes the possibility to support the development team in the specification of new ML solutions by recommending existing specifications.

To address the [Design Challenge \(CH-1\)](#), there is a need for concepts and methods that introduce formalism to ML solution specifications. Through formalization, ML solution can be processed by software systems. There also needs to be a defined way of collaboration for all roles in the development team. Similarly, concepts and methods need to offer a full trace of domain-specific goals and requirements and how they lead to concrete software components. This allows a better integration of the new ML solution with systems available at the use case.

1.3.2. Configuration Challenge

The second challenge pertains to the configuration step in the ML solution development process. Data scientists and software developers build different ML solutions with different ML algorithms, training settings, hyperparameter values, data preprocessing techniques and IT resources. The search space, i. e., all combinations with the elements mentioned above, quickly becomes too large to be explored exhaustively (Xin et al., 2021). Yet, software developers and data scientists typically try every tool combination they have available (Zaharia et al., 2018). This opens the risk of incurring in many training iterations in search for a better performing ML solution. Moreover, as the number of iterations grows, it becomes increasingly difficult to determine the exact combination of software components and settings used to build a good performing ML solution. This problem is exacerbated if there is no standard procedure to evaluate the ML solution's performance. It is thus important to be systematic during this highly creative step of the development process.

To address the [Configuration Challenge \(CH-2\)](#), there is a need for concepts and methods to ensure comprehensive documentation of ML solution configurations. This documentation should ensure the complete reproducibility

of ML solutions. In this context, reproducibility covers the observed performance and the software component configuration. Data scientists and/or software developers should be able to rebuild an ML solution based on its documentation and obtain the same performance as documented. Moreover, as more ML solutions are developed, it is important that this documentation has a systematic format to allow the concentration of all documents in a single source.

1.3.3. Selection Challenge

The third challenge pertains to the selection step in the ML solution development process. Decision makers review the candidate ML solutions developed in the previous step to choose which alternative is most suitable for the use case. However, use case suitability depends on multiple criteria, e. g., development and deployment costs, execution time. Not all of these criteria are reported through ML metrics. Decision makers have difficulties selecting ML solutions because of the expert knowledge required to understand them (Baier et al., 2019). For instance, they may be confronted with ML metrics, e. g., RMSE or F1 Score, whose scale and values are of little utility without expert interpretation. This is particularly problematic when the ML tools themselves offer little information about the trained ML model (Xin et al., 2021).

To address the [Selection Challenge \(CH-3\)](#), there need to be concepts and methods to facilitate understanding and comparing ML solutions for non-experts. These methods should allow domain experts and decision makers to understand the overall performance of an ML solution beyond single ML metrics. Additionally, the methods should make it easy for non-experts to understand the performance trade-offs of choosing one ML solution over others. As the number of ML solutions increases, it becomes important to be able to compare multiple ML solutions in a reasonable time.

1.3.4. Process Challenge

The fourth challenge pertains to the overall ML solution development process. The interaction of all roles needs coordination to carry out all steps. Coordination implies clearly-defined responsibilities and step sequences. Without clear responsibilities, different roles working together, e. g., in the design step, may duplicate activities. Another possibility is that by not involving a certain role, e. g., the domain expert, some activities are omitted and their absence noticed only much later. Moreover, performing the three steps in the development in exclusive sequence may become very inefficient for large scale projects where some ML solution components will be finished before others.

To address the [Process Challenge \(CH-4\)](#), there need to be concept and methods to organize the sequence of activities and steps in the development process. The methods should determine the complete list of activities that every role must perform to complete every step. This list of steps should ensure not only that development is complete and time-efficient, but also that it is reproducible and that domain-specific concerns are satisfied. Methods should also devise the correct sequence of activities to eliminate unnecessary iterations and to parallelize some activities to reduce overall development time. Moreover, there has to be a proper documentation of the progress throughout the process, so that the resulting ML solution is reproducible and and its performance understandable. This can help establish best practices that can later be applied on any new ML solution development project.

1.4. Research Contributions

This section presents four research contributions to address the research challenges introduced in [Section 1.3](#). [Figure 1.6](#) shows the research contributions across the typical ML solution development process to indicate the challenges they address. Preliminary versions of the research contributions have been published in different proceedings of conferences and in a journal (Villanueva Zacarias et al., [2020](#); Villanueva Zacarias, Ghabri,

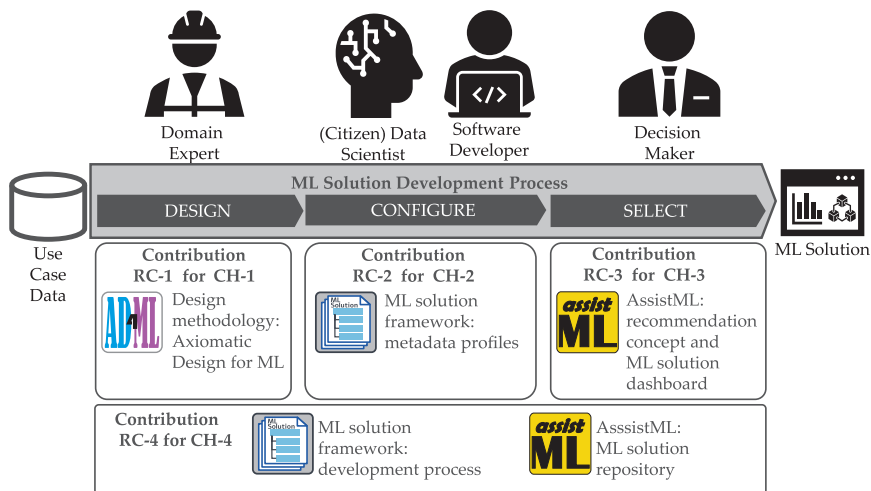


Figure 1.6.: Contributions to address the challenges in the development of ML solutions.

et al., 2021; Villanueva Zacarias, Reimann, et al., 2018; Villanueva Zacarias, Weber, et al., 2021; Villanueva Zacarias. et al., 2017). They have been edited and expanded to be included in this dissertation. All contributions taken from these publications and described in this dissertation are the original work of the dissertation author. Additionally, software implementations of these contributions have been made publicly-available online ¹.

To obtain these contributions, research was conducted following the design science approach of Hevner et al., 2004. This research methodology is suitable for contexts where people, organizations and technology interact. Design science proposes the creation of research artifacts as a way to know and solve a certain problem domain. Research artifacts can be a construct or concept, a model, a method or an instantiation (Hevner et al., 2004). These artifacts are developed iteratively in build-and-evaluate cycles. In each cycle, knowledge is applied on an artifact, and the artifact is evaluated in a practical use case. This research thus develops concepts and methods

¹<https://github.com/al-villanueva>

as contributions for each identified research challenge. The research then assesses their effectiveness with prototypical implementations on concrete use cases, i. e., instantiation artifacts in the context of design science.

1.4.1. Research Contribution for the Design Challenge

The design methodology [Axiomatic Design for Machine Learning \(AD4ML\)](#) addresses the design challenge [CH-1](#). AD4ML adapts the Axiomatic Design methodology (Suh, 2001) to enable the specification of ML solutions. ML solution specifications made with AD4ML are based on a mathematical formalism that allows them to be processed by software systems. They are also built in a modular manner, which facilitates coordination and collaboration of the development team and increases their reusability. This contribution enables the systematic consideration of domain-specific requirements from the very beginning of ML solution development. It also enables the documentation of ML solutions at a domain-specific level in order to trace the function of each solution component to a domain-specific requirement. This contribution is composed of the following sub-contributions:

Requirements (RC-1.1) A set of feasibility requirements for the design of ML solutions in manufacturing.

Concept (RC-1.2) AD4ML – An adaptation of Axiomatic Design to specify ML solutions.

Implementation (RC-1.3) Prototypical implementation of AD4ML as a REST service available via a web interface.

1.4.2. Research Contribution for the Configuration Challenge

The ML solution framework addresses the configuration challenge [CH-2](#). The ML solution framework offers a comprehensive yet flexible format to collect all information needed to develop and reproduce ML solutions. The framework includes information about the domain-specific task and preferences, the use case data properties, hardware and software configurations, ML

algorithm and training settings, and the observed performance. The comprehensiveness of the metadata collected makes it possible to standardize the documentation of ML solutions with considerably different configurations. For instance, it can be used to document ML solutions developed with visual tools, e. g., KNIME or RapidMiner, ML solutions with ML models for regression, binary classification or multi-class classification. This contribution improves the reproducibility of ML solutions by documenting more than the ML model and its settings. This improves the ability of development teams to share their results with each other. This contribution is composed of the following sub-contributions:

Requirements (RC-2.1) A set of requirements regarding the combination and configuration of ML solution components.

Concept (RC-2.2) A conceptual ML solution framework consisting of four metadata profiles.

Implementation (RC-2.3) Prototypical implementation of metadata profiles for exemplary use cases as well as a visualizer.

1.4.3. Research Contribution for the Selection Challenge

AssistML addresses the selection challenge [CH-3](#). AssistML is a recommendation concept to find suitable ML solutions that can be adapted to be reused in new use cases. With AssistML, users can easily search for and compare many ML solutions without the need of expert knowledge. The concept takes into account performance preferences and presents users, i. e., citizen data scientists and decision makers, intuitive reports of each candidate ML solution. The reports they obtain from AssistML allow users to understand the performance trade-offs of employing each candidate and select the ML solution that best meets the use case needs. This contribution improves the ability of development teams to implement ML solutions for new use cases by reducing the iterations needed to find suitable solution components or configurations. It also empowers citizen data scientists to take up development projects on their own, thus reducing the demand for expert data

scientists in the company. This contribution is composed of the following sub-contributions:

Requirements (RC-3.1) A set of requirements regarding the complexity of recommending reusable ML solutions to non-expert users.

Concept (RC-3.2) [Assisted Machine Learning \(AssistML\)](#) – A concept to recommend and select ML solutions with consideration of performance preferences.

Implementation (RC-3.3) Prototypical implementation of AssistML and the metadata repository.

1.4.4. Research Contribution for the Process Challenge

The development process of the ML solution framework together with a metadata repository address the process challenge [CH-4](#). The development process determines a comprehensive list of activities for the design, configuration and selection steps. It guides all members in the development team to carry out their activities in the right order and ensures that reproducibility, standardization and good practices are in place.

Because the development process spans over many activities and involves different roles, any progress made, e. g., implemented software, preprocessed data and domain-specific decisions, has to be properly documented. Proper documentation of every activity allows stakeholders to understand and reproduce the results obtained throughout the development process. A metadata repository was conceived to address this aspect of CH-4. The repository is designed to collect and organize metadata of relevant intermediate results during the development project. These include the manner in which data is preprocessed, the settings used to train the ML model(s), the performance evaluation procedure followed as well as the performance scores obtained. This metadata repository is conceived to document ML development projects regardless of the technology stacks used to allow the consistent documentation of diverse ML solutions. This also allows the meta-

data repository to serve as data foundation for the recommendation concept AssistML.

This contribution is composed of the following sub-contributions:

Concept (RC-4.1) A conceptual guidance process to develop ML solutions.

Concept (RC-4.2) An extensible, technology-agnostic metadata repository to document progress during the ML solution development process.

Implementation (RC-4.3) A metadata repository to concentrate all documentation, source code, and data to reproduce the ML solution developed, explain its performance and serve as basis to recommend ML solutions for new use cases.

1.5. Dissertation Outline

This dissertation is organized in five logically ordered chapters.

[Chapter 2 – Theoretical Background](#) gives an overview of the concepts and theories that underpin the contributions of this dissertation. This includes an introduction to data-driven machine learning, production systems, applications of ML in manufacturing and existing approaches to the design, configuration and selection of ML solutions.

[Chapter 3 – ML Solution Framework](#) discusses the ML solution framework and ML solution development process. These contributions structure the complete ML solution development project. They also form the basis for the contributions presented in the following chapters, i. e., AD4ML in [Chapter 4](#) and AssistML in [Chapter 5](#). This chapter proposes these contributions in response to a set of requirements extracted from a literature review. The chapter also describes the application of the framework on an exemplary use case for end-of-line fault detection. The chapter closes with a prototypical implementation of the metadata profiles.

[Chapter 4 – Axiomatic Design for Machine Learning](#) discusses the use of the Axiomatic Design methodology to specify ML solutions. The chapter first discusses the requirements that the specification of ML solutions for

manufacturing use cases imposes on any design methodology. It then introduces a reference use case to serve as a guiding example for the adaptations that constitute AD4ML. Afterwards, the chapter discusses further relevant aspects of using AD4ML to specify ML solutions. Concretely, it discusses a visualization method for AD4ML specifications, methods to assess the quality of design decisions and overall AD4ML specifications, and an approach to carry out the specification with AD4ML following agile principles.

Chapter 5 – AssistML discusses AssistML, a concept to recommend existing ML solutions for new predictive use cases. The chapter first determines the requirements that the concept must fulfill to support its target users, i. e., the citizen data scientist and the decision maker. It then introduces the metadata repository as a key requirement for the AssistML recommendation process. The chapter discusses the steps of the recommendation process in detail and evaluates its ability to recommend useful ML solutions with two exemplary use cases. The chapter closes with a prototypical implementation of the recommendation process and the metadata repository.

Chapter 6 – Conclusion and Future Work summarizes the contributions presented in this dissertation and discusses the extent to which they address the research challenges from **Section 1.3**. The chapter closes with an overview of the directions in which the research can continue. A list of publications, supervised theses and projects as well as publicly released software can be found at the end of the dissertation.

CHAPTER



THEORETICAL BACKGROUND

In order to delimit the scope, this chapter introduces concepts related to the main research contributions. The first section discusses the process of designing new production systems, as this is the context in which the specification of ML solutions takes place. The second section describes the characteristics of data collected or generated within such production systems to determine the data sets that can be expected to be used for machine learning. Then, the third section characterizes Machine Learning and discuss the common applications of this kind of ML in manufacturing use cases. The last three sections give a brief overview of existing approaches to design, configure and select ML solutions that serve as context for the contributions presented in [Chapters 3 to 5](#).

2.1. Characteristics of Production System Design

This section first describes production system design in the context of the product conception phase according to Eigner (Eigner and Stelzer, 2009) and the Smart Engineering initiative by Acatech (Anderl et al., 2012). Afterwards, three elements of production system design are further detailed.

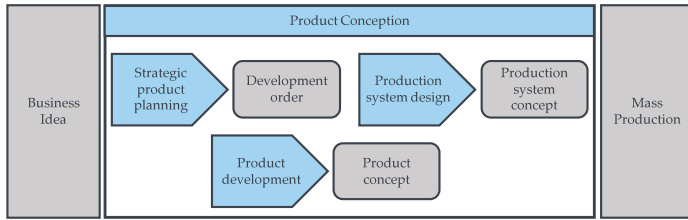


Figure 2.1.: Overview of the product conception phase. After descriptions from (Anderl et al., 2012) and (Eigner and Stelzer, 2009)

These are (1) classification systems (Grabowski et al., 2002), (2) feature technology (Haasis et al., 2003), and (3) model-based systems engineering (Eigner, 2013).

2.1.1. Production System Design within the Product Conception Phase

Product conception is part of the product life cycle (Eigner and Stelzer, 2009). It covers processes to develop a business idea into viable product and production system concepts (see Figure 2.1). Using these concepts, the mass production phase can be launched.

The activities in product conception are grouped into three processes: the *strategic product planning*, the *product development* and the *production system design*. These processes do not occur in isolation. They interact with each other as their corresponding results get further detailed. In this sense, they do not follow a traditional process sequence (Anderl et al., 2012). The goal of *strategic product planning* is to ensure the economic viability of the new product. This requires verifying the existence of a business need, business models, a market segment and the potential for the company (Anderl et al., 2012). During *product development*, the product is conceived from mechanical, electrical and electronic, and software standpoints. The resulting product concept contains exact and binding specifications of the product's technical functionality, costs, and quality levels (Anderl et al., 2012). As for the *production system design*, the goal is to determine how to manufacture the product according to its product concept. For this purpose, the product

concept is broken down into manufacturing operations. Each manufacturing operation is responsible for a product characteristics or functionality. Manufacturing equipment, facilities, and resources are assigned to each manufacturing operation (Anderl et al., 2012). Depending on the viability to manufacture the product, either the product concept or the production system concept can be adapted to better suit each other.

The product conception phase ends with complete and viable specifications of the product and the production system. The product concept is a complete digital description of the product, referred to as Digital Master (Eigner and Stelzer, 2009), Virtual Product or Virtual Prototype (Anderl et al., 2012). The production system concept, also referred to as Digital Factory or Virtual Production (Anderl et al., 2012) contains, e. g., the work plan, the layout for the production and assembly lines, and the design of working stations.

The need for interdisciplinary efforts to develop both product and production system concepts is satisfied through *Cross Enterprise Engineering* (Eigner and Stelzer, 2009). This concept implies cooperation and coordination across disciplines, organizations and systems spanning throughout product conception and the later phases of the product life cycle (Eigner and Stelzer, 2009).

2.1.2. Classification Systems

The design of production systems considers a wide range of system components. Components include manufacturing equipment, manufacturing operations, production facilities, resources, and production technologies. Their technical descriptions are stored in different data sources (Grabowski et al., 2002). These can include reports, specification sheets, and test results. Besides, each component has a distinct relationship with the rest of the production system. These relationships are also kept in data sources. For instance, a specification sheet can describe the type of material that a machine can handle, thus linking the material to the machine. Designing a production system without an organizing structure impacts the process

efficiency. This is due to the extra effort to organize and find data (Grabowski et al., 2002). The possibility to reuse specifications is also reduced.

Classification systems provide a structure for components in a production system. They assign each component a compact classification code that describes its key properties. Examples of classification systems are eCl@ss, the European Article Numbering *EAN* and North American Industry Classification System *NAICS* (Grabowski et al., 2002).

Classification codes are built out of a hierarchy of key properties. Each key property is a level in this hierarchy, its values encoded with a few letters and numbers (Grabowski et al., 2002). The combination of key property values uniquely identifies a component with a standard, meaningful representation. For example, a classification system can be built using three key properties. The first property can distinguish the type of system component. The second property can state the product line in which the component is used. The third property can identify the revision number of that component.

Without a classification system, systems component can only be found via search techniques. This implies issuing custom search queries that often produce many slightly relevant results. Besides, it is not guaranteed that the query finds the desired component (Grabowski et al., 2002).

2.1.3. Feature Technology

Feature technology was introduced at the beginning of the 2000's by Daimler Chrysler and Dassault Systèmes to address the need to integrate data about the product, the production system and the resources used for production (Haasis et al., 2003). Features are uniform carriers of descriptive and semantic information to ensure a continuous dataflow throughout the product life cycle. Depending on the kind of data they capture, they can have one of four different types: *design*, *manufacturing*, *assembly* and *inspection*.

The use of feature technology in product conception begins with the design of product parts. A product part is composed of design features. Each design feature has a specific function associated to it. This allows designers to choose features according to the functions that the new part should have

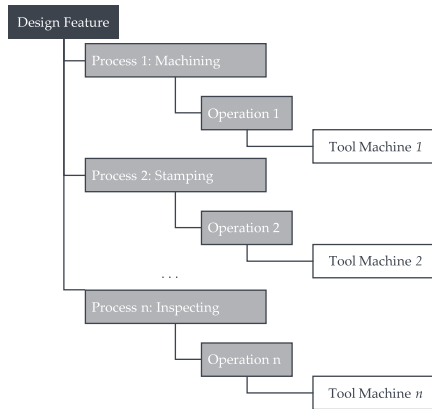


Figure 2.2.: Feature tree showing the connection between part feature, process and resources. After (Haasis et al., 2003)

(Haasis et al., 2003). This promotes standardization of design features, which in turn results in cost efficiencies.

During production system design, design features are mapped to manufacturing and inspection processes. This is called feature mapping and is shown in Figure 2.2. Afterwards, inspection processes are mapped into specific operations. Then, these operations are linked to the machines capable of doing them (Haasis et al., 2003).

Thanks to this linking from part to machine, the production costs, machine capacities, and supply logistics can be more accurately estimated. This end-to-end linkage also offers the possibility to trace any changes to either the product, the production system or the machines.

2.1.4. Model-based Systems Engineering

With products incorporating more electronic and software functionality, there has been a growing need to make product conception more interdisciplinary. In response to this need, a new discipline called Systems Engineering was born. In Systems Engineering, both the product and the production system are considered systems as they are "sets of elements (...) that accomplish a

defined objective." (Walden et al., 2015). Moreover, their interactions make it necessary to consider them parts of a *system of systems* that is designed during the product conception phase.

Model Based Systems Engineering (MBSE) is an approach in systems engineering that aims to develop systems through the construction and refinement of a unified, coherent, digital system model (Eigner, 2013; Ramos et al., 2011). The system model, its constituent models and model elements are developed using appropriate modeling languages and modeling tools, e.g., SysML in Artisan Studio (Ramos et al., 2011). The resulting models are stored and managed in a model repository (Ramos et al., 2011). The models in the repository become a single coherent data source for all activities in product conception and for later phases in the product lifecycle. Models describe systems, i.e., the product and production system, in a formal, complete and consistent manner.

In contrast to models, documents change the textual specifications as they are exchanged throughout the product conception. Furthermore, models contain interconnected information about the requirements, structure, behavior and parameters of the systems (Ramos et al., 2011). The interconnections are made possible thanks to a data model supporting hierarchies, cross-references between model aspects and typed connections within the models (Eigner, 2013).

2.2. Data in Manufacturing Use Cases

The widespread adoption of software systems throughout the production system has grown over the years (Tao et al., 2018). This includes information systems to manage aspects of the product lifecycle, e.g., **Customer Relationship Management (CRM)**, **Manufacturing Execution System (MES)**, **Enterprise Resource Planning (ERP)**, **Supply Chain Management (SCM)** and **Product Data Management (PDM)** systems. Also included are software systems to handle product data, e.g., **Computer Aided Design (CAD)**, **Computer Aided Engineering (CAE)** and **Computer Aided Manufacturing (CAM)**

systems. All these systems generate data and store them using their own databases or data formats for all members of an organization to access (Tao et al., 2018). Recently, the amount of data to analyze the production system has grown significantly. This is due to the use of digital sensors at production system facilities, the access to open data sets online and to customer data from social media channels (Tao et al., 2018). Therefore, distributed data storage systems, e. g., data lakes or cloud services, are required to store the larger data sets (Giebler et al., 2019).

This has led organizations to cope with the challenges of big data (Gandomi and Haider, 2014; Grover and Kar, 2017; Moynes and Iskandar, 2017; O'Donovan et al., 2015; Tao et al., 2018). Besides the sheer size of collected data, other characteristics have been adopted to define big data. These characteristics have been summarized in five V's, namely *volume*, *variety*, *velocity*, *veracity* and *value* (Gandomi and Haider, 2014; Grover and Kar, 2017; Moynes and Iskandar, 2017). Each of these characteristics indicate potential challenges that industry practitioners need to address to use their data in the development of ML solutions.

Volume refers to the magnitude of the data relative to the available capacity storage (Gandomi and Haider, 2014; Grover and Kar, 2017). This characteristic is mainly relevant to assess the scalability of the data storage in the production systems. For instance, the addition of sensors in the production system can have a big impact in the amount of data storage needed.

Variety refers to the different data types collected in the production system (Gandomi and Haider, 2014). This characteristic is of particular relevance in production systems given the high heterogeneity of data sources, which go from traditional databases of MES or ERP systems (Grover and Kar, 2017; Moynes and Iskandar, 2017) to semi-structured or unstructured data of Internet of Things (IoT) sensors. Currently, the most common industry data types are structured data, e. g., contained in relational databases, time series data, and unstructured text (Mäkinen et al., 2021). Assessing this characteristic helps to determine the type of preprocessing needed to analyze the data and obtain meaningful information from it (Grover and Kar, 2017).

For example, unstructured text requires some type of representation before being used by an ML model. Representations include n-grams, bag of words or word embeddings (Sebastiani, 2002).

Velocity refers to the rates at which data is generated, collected, analyzed or acted upon (Gandomi and Haider, 2014). This characteristic is reflected in production systems in the execution models of different software systems. Some software systems may process data in batches, e. g., at the end of a production shift, whereas others process data in real-time as products advance the assembly line. This characteristic is specially challenging for data scientists (Grover and Kar, 2017), who need to take into account the different speeds of data sources for their preprocessing.

Veracity refers to the reliability of data sources, i. e., how correct the facts stated by the data source are (Gandomi and Haider, 2014). This characteristic is closely related to the assessment of the *data quality*, i. e., assessing how accurate, complete, truthful, or objective a data source is (Grover and Kar, 2017; Moyne and Iskandar, 2017). The higher the data quality, the better is the veracity of the data source. This is of particular importance in production systems to assess customer data or unstructured text data where judgments are made upon, e. g., social media excerpts or quality assurance reports. Assessing this characteristic and having good mitigation strategies to handle bad quality data is critical to increase the adoption of ML solutions (Moyne and Iskandar, 2017).

Value refers to the degree of usefulness of a data source based on the insights that can be extracted from it (Gandomi and Haider, 2014; Grover and Kar, 2017). In general, big data is expected to have low-density value, i. e., that an individual data point is of little utility compared to the analysis of many data points (Gandomi and Haider, 2014). An alternative stance is that from a large data source, analysis has to be carried out to extract the valuable data that can be acted upon (Grover and Kar, 2017), e. g., extract a few representative performance or configuration patterns from a big data set. Taking this characteristic into account is important as it is the value extracted from data by ML solutions that justifies the investments made to develop them.

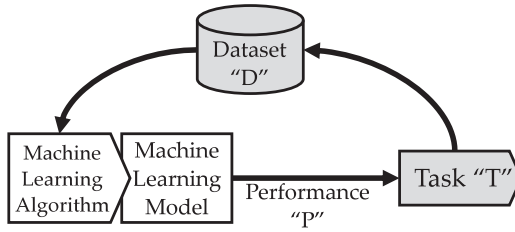


Figure 2.3.: General operation of Machine Learning

2.3. Machine Learning

Machine Learning (ML) is a subfield of artificial intelligence that aims to develop algorithms that learn and improve their performance in the execution of a specific task (Kashyap, 2018; Mitchell, 1997). The general operation of Machine Learning is shown in Figure 2.3. A machine learning algorithm identifies and learns patterns contained in a data set D regarding a given learning task T . These patterns indirectly represent knowledge which is generalized in a *trained model* following the ML algorithm logic. The trained model performs the task with a certain performance P (Mitchell, 1997), which can be measured using different metrics, e.g., accuracy for supervised learning tasks. This learning task consists of making an inference for an unseen case based on the learned patterns from the data. Depending on the type of learning task, the algorithms to train models and make inferences vary. This includes supervised learning, unsupervised learning, reinforcement learning and association rules (Alpaydin, 2009). This dissertation focuses on supervised learning.

In supervised learning, the data set D is composed of input features X and a class data feature Y . A supervised learning algorithm trains a model $m(X|\theta)$ that makes inferences or predictions Y' based on the input data features X and the learning parameters θ (Alpaydin, 2009). The model thus aims to map certain values of X to values of Y' , so that the error between Y' and Y is minimized.

Depending on the data type of Y , i. e., the type of prediction, supervised learning tasks are subdivided in classification, for categorical data, or regression, for numerical data (Alpaydin, 2009). The specific learning task determines the type of learning algorithms that can be applied to it. Representative examples include Decision Trees, Random Forests, Naive Bayes, Logistic Regression, Support Vector Machines and Artificial Neural Networks. Each learning algorithm has a set of assumptions that make use of certain properties in the input feature X . For instance, certain variants of the Decision Trees algorithm rely on the concept of gini impurity to determine which data feature from X can classify all observations in the data set best. (Kotsiantis, I. Zaharakis, et al., 2007) and (Kotsiantis, I. D. Zaharakis, et al., 2006) give a comprehensive overview of established learning algorithms.

ML models can be further divided depending on the types of labels they produce. Some models produce single class predictions, i. e., a class out of two or many is selected for each observation in X . ML models trained with probabilistic algorithms produce prediction probabilities (Bishop, 2006). Finally, ML models can produce multiple simultaneously correct class predictions, i. e., produce a multi-label output (Kubat, 2017).

Supervised learning algorithms have been widely used to support production processes since the early 2000s, as a review of Wuest et al., 2016 shows. The technology is applied by developing a Machine Learning (ML) solution for a concrete use case in the production process.

A ML solution denotes the set and sequence of software components and algorithms necessary to deliver a predictive capability needed at a use case. As shown in Figure 1.1, a ML solution has to include (1) software components to read data from relevant data source(s), (2) software components to preprocess data, (3) one or more machine learning algorithms to train a machine learning model, e. g., a decision tree, and (4) postprocessing components to communicate the prediction to the use case. Furthermore, a ML solution includes (5) software components to deploy the machine learning model in the use case to make predictions. Finally, the ML solution also includes the (6) computing resources all the previously mentioned software components can operate on.

From the perspective of feature technology, ML solutions are *machines* performing inspection operations (Sharp et al., 2018). They are, for instance, deployed at test stations as part of an intelligent quality control system. They can trigger triage operations in an assembly line to send potentially defective assemblies into a more detailed inspection. Moreover, they can be part of monitoring dashboards to provide warnings of potential machine outages.

Depending on the specific use case in which it is deployed, the ML solution needs to be integrated with different systems. Just like regular machines and equipment, the linking of ML solutions with specific inspection operations has to be reflected in the production system concept.

2.4. Approaches to Design ML Solutions

The first step in the development process has the goal of specifying the ML solution to be developed (see [Section 1.2.3](#)). For this purpose, different design methodologies have been adapted or proposed to partially address this step.

The most prominent methodology is the [Cross Industry Standard Process for Data Mining \(CRISP-DM\)](#) (Chapman et al., 2000; Shearer, 2000). [CRISP-DM](#) is a high-level reference process to manage data mining projects in industry (Shearer, 2000). It is widely adopted by industry practitioners to develop data mining, data analytics, business intelligence and ML projects. [CRISP-DM](#) covers the whole development process in six phases, the first one of which addresses, among other things, the design of the ML solution. The result of this first phase is the data mining problem definition which, along with a project plan, outlines the project scope and goals (Chapman et al., 2000).

The [Quality Function Deployment \(QFD\)](#) methodology (Akao and King, 1990) has been adapted to design ML solution-like systems. [QFD](#) was originally conceived to design complex products based on customer preferences, the so-called *voice of the customer*. Based on an iterative mapping between requirements on one side and means to fulfill them on the other, [QFD](#) deduces

the product and production process specifications from customer requirements (Crowe and Cheng, 1996). This methodology has been adapted for software development (Herzwurm, Schockert, et al., 2015) and more specific application domains, e. g., cloud computing (Herzwurm, Pietsch, et al., 2012).

Other methodologies have been proposed to deal explicitly with the design of ML solutions. One is the Analytics Canvas (Kühn et al., 2018). This methodology is meant to facilitate communication and planning before the development process. The canvas offers a one-page-long overview of the IT infrastructure, data and stakeholders required for a concrete use case (Kühn et al., 2018). Another methodology is the conceptual modelling framework by Nalchigar and Yu, 2020. The framework aims to model an ML solution-like system using three modeling views or sub-models. These are the *business*, *analytics design* and *data preparation* views. Each view corresponds to a specific group of team members, i. e., business analysts, data scientists and data engineers, who are responsible for building and updating them (Nalchigar and Yu, 2020). Contrary to the Analytics Canvas, the modeling framework sacrifices visibility of the resulting model for a comprehensive specification based on predefined meta-models.

2.5. Approaches to Configure ML Solutions

The second step in the development process refers to the assembly of data, software components, hardware and software resources, and ML algorithms and their configuration to implement different ML solutions (see Section 1.2.3). Given the explorative and iterative nature of this step, there is a risk to follow brute-force and/or trial-and-error approaches, i. e., the development team tries all available implementation options (Zaharia et al., 2018). Configuration approaches thus are needed to provide structure to this step's iterations.

There are two methodologies that guide the process of combining data and solution components. First, [Knowledge Discovery in Databases Process \(KDD\)](#)

is another established methodology (Fayyad et al., 1996). **KDD** structures the process of building ML solutions in five steps. Through these steps, data is selected, preprocessed, and transformed to make it accessible to data mining techniques. Then, data mining techniques, among them ML algorithms, can be applied to extract patterns, i. e., useful observations that can be applied to new data with some degree of certainty (Fayyad et al., 1996). The process ends with an evaluation of all extracted patterns. In this sense, the **KDD** process organizes ML solution configuration around data exploration rather than domain-specific goals. Second, is the process developed by the SAS institute: **Sample, Explore, Modify, Model, Assess (SEMMA)** (Azevedo and Santos, 2008). Its goal is to guide users through the exploration of data samples to build and assess an ML model. **SEMMA** focuses on the data analysis, data preprocessing and modeling aspects of the solution configuration. Thus, it leaves aside domain-specific or deployment aspects, e. g., performance preferences or deployment constraints.

Besides these methodologies, other approaches have been conceived to track the ML solution configuration. These approaches offer a common data structure or platform to document every relevant characteristic of the developed ML solutions. ML Model Scope supports deep learning models from different software libraries on different operative systems and hardware platforms (Li et al., 2019). MLflow is a platform for tracking the parameter values and resulting performance of ML models via a REST API (Zaharia et al., 2018). OpenML is designed to be a repository of data sets and the ML models that are trained on them (Vanschoren et al., 2014). Its goal is to enable comprehensive comparisons between ML models built for the same data set.

2.6. Approaches to Select ML Solutions

The third step in the development process focuses on the comparison of different ML solutions to determine the most suitable ones for deployment (see [Section 1.2.3](#)). Currently, the comparison is made based on ML metrics com-

puted on benchmark data sets (Wagstaff, 2012). In this context, Bourrasset et al., 2019 have proposed a series of metrics for enterprise benchmarks. They propose metrics to benchmark model training, hyperparameter optimization, and the execution time for an ML to make predictions. Yet, their proposed metrics have not been completely covered by existing benchmarking approaches (Bourrasset et al., 2019).

Other ML Benchmarking approaches devise standard evaluation approaches, rank ML solutions based on the measured values, and assign scores to each implemented ML solution. MLPerf (Mattson et al., 2020) is an industry-led benchmarking platform that defines reference tasks, metrics, and categories to compare deep learning models using different ML software libraries and hardware architectures. DeepOBS (Schneider et al., 2019) is a benchmarking suite exclusively tailored for deep learning optimizers using TensorFlow.

ML SOLUTION FRAMEWORK

A team tasked with the development of ML solutions faces difficulties related to the *configuration* (CH-2) and *process challenges* (CH-4) (see [Section 1.3](#)). As shown in [Figure 3.1](#), the team may start the development with access to use case data and a domain-specific problem definition. Yet, there is no systematic methodology to structure the interactions between domain experts and other team roles in a development team to configure suitable ML solutions for their particular problem (H. H. Hoos et al., 2017; Pham and Afify, 2005; Wuest et al., 2016). Furthermore, ML solutions are made of several software components, which are difficult to understand by domain experts and decision makers. Specifically, these roles lack adequate documentation to comprehend and compare different solutions and to finally select the solution that matches their requirements. This challenge is further aggravated given the need to be able to reproduce the ML solution and its measured performance later in a productive environment.

This chapter proposes a framework to develop ML solutions based on a process that systematically profiles solution components. Domain experts can define their domain-specific problem in terms they are familiar with, e. g., in terms related to quality control of production. The ML solution framework

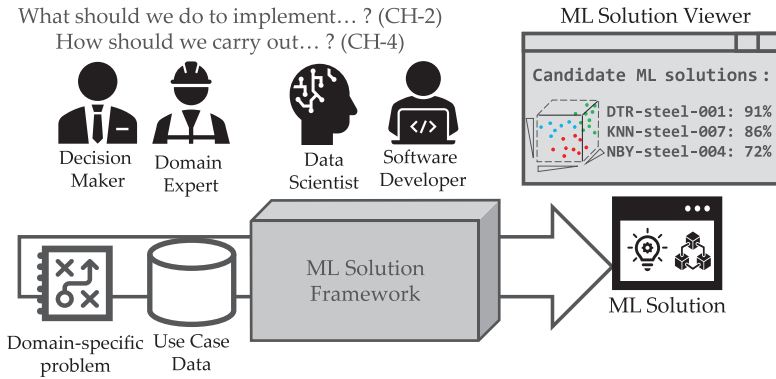


Figure 3.1.: General contribution: Development teams can systematically develop and review ML solutions built for their data and use case.

structures the development of suitable ML solutions and contributes to their reproducibility. In the end, the framework presents the domain experts with the resulting ML solutions using concise and relevant information. Users can compare the ML solutions taking into account their individual needs via a visual, easy-to-use ML solution viewer. Altogether, the ML solution framework provides a systematic approach that brings transparency in a situation that is highly-technical for domain experts.

The contents of this chapter are an edited version of a separate publication (Villanueva Zacarias, Reimann, et al., 2018). All concepts extracted from it and used in this dissertation are the original work of the dissertation author.

This chapter is organized as follows: [Section 3.1](#) discusses related work and current limitations with respect to the development of ML solutions in manufacturing. [Section 3.2](#) derives five design requirements that a framework supporting the development of ML solutions has to achieve. [Section 3.3](#) introduces the framework components and discusses how each of these components contributes to the fulfillment of the design requirements. [Section 3.4](#) presents a prototypical implementation of the solution framework

and evaluates it on the basis of representative use cases. Finally, [Section 3.5](#) concludes and lists possible future work.

3.1. Related Work

This dissertation surveyed related work in three research areas: (1) challenges in applications of machine learning (ML) algorithms in manufacturing, (2) empirical evaluation of ML algorithms, and (3) automated machine learning.

3.1.1. Challenges of ML Solutions in Manufacturing

Langley and Simon are among the first to list machine learning applications from several domains (Langley and Simon, [1995](#)), with the oldest one in manufacturing dating back to 1985. According to them, the main challenge for a successful use of ML is to specify a correct domain problem formulation, regardless of the algorithms used.

A decade later, Pham and Afify, [2005](#) confirm the importance of problem formulation and recognize two additional challenges: firstly, the need for techniques to handle bigger amounts and more varied types of data and, secondly, the need to combine multiple algorithms in order to increase the quality of an analytics solution (Pham and Afify, [2005](#)). Recently, Wuest et al. (Wuest et al., [2016](#)) have verified these challenges to be even more important now. One consequence is a high complexity for acquiring, understanding, and preparing data as needed by the desired analytics solution. Furthermore, it is now very difficult to select suitable algorithms from the many available ones and to properly configure them. For classification problems alone, candidate algorithms can include Decision Trees (logic-based), Naive Bayes (statistical learning), k-Nearest Neighbors (instance-based), Deep Learning (perceptron-based), Support Vector Machines, as well as the sub-variants and combinations thereof (Kotsiantis, I. D. Zaharakis, et al., [2006](#)).

A work in recent years handled the above-mentioned challenges with an established methodology, offering guidelines throughout the process (Lieber et al., 2013). Yet little detail is given on how a particular algorithm is selected and how its parameters are configured. Only low-level ML evaluation aspects (cross validation or feature selection) are taken into consideration. This not only makes the comparison of techniques difficult for domain experts, it also neglects other relevant issues, such as the practical feasibility or execution costs, which should play a role in the decision.

3.1.2. Empirical Evaluation of Machine Learning Algorithms

The evaluation of different ML algorithms on the same or multiple data sets is a common practice to identify appropriate algorithms for a certain kind of problem or for certain kind of data. Several studies have been carried out for this purpose (Demšar, 2006; Dogan and Tanrikulu, 2013; Olson, La Cava, et al., 2017; Sokolova and Lapalme, 2009). Such an evaluation of algorithms is usually done based on predefined performance metrics, e. g., accuracy or precision. While these metrics constitute the undeniable baseline for ML research, they do not directly reflect the actual goals domain experts pursue when applying ML. For instance, these domain experts are rather interested in reducing costs or increasing quality of their products.

3.1.3. Automated Machine Learning

A novel research area that is related to the ML solution framework is automated machine learning or AutoML (Feurer, Klein, et al., 2015). Its main goal is to automatically generate and configure ML solutions through empirical testing under predefined conditions on a target data set. Furthermore, it aims to leverage human expertise by allowing users to restrict the types of algorithms, as well as the performance metrics to evaluate candidate algorithms (H. H. Hoos, 2008).

The framework proposed in this chapter focuses on two major tasks of AutoML: (1) *algorithm selection*, i. e., which type of algorithm fits the appli-

cation problem, and (2) *parameter tuning*, dealing with the question of how algorithm parameters should be configured to achieve satisfactory results. Various methods have been proposed for both tasks (Bischl et al., 2016; Hutter et al., 2009; Snoek et al., 2012). However, the area of AutoML still faces several challenges. In particular, algorithm selection and parameter tuning considering multiple objectives is a tedious and sophisticated problem. Furthermore, related work lacks systematic methodologies to guide the selection and configuration of algorithms for concrete real-world use cases (H. H. Hoos et al., 2017).

3.2. Design Requirements

In response to the challenges discussed above, the research derived a set of design requirements for a framework to assist in the configuration and selection of ML solutions. These requirements concern (1) the specification of a domain-specific problem, (2) the efficient and accurate data utilization, (3) the consideration of available IT resources, (4) the property-based and performance-based configuration and selection of ML algorithms, and (5) the enhanced ML solution comprehensibility.

3.2.1. Specification of a Domain-specific Problem Definition

The framework needs to enable domain experts to specify the problem or task to be solved by analytics in terms they are familiar with. For instance, quality engineers should be able to literally define their problem such as: *Identify steps of the assembly line that cause severe quality issues and most of the rework for products A and C.* This allows domain experts to abstract from all technicalities of the formal optimization problem of finding and configuring adequate ML solutions. Instead, they can focus on the domain-specific goals and principles to be fulfilled, as well as on the priorities between possibly conflicting goals (e. g., quality of the results vs. processing time). Once established, this set of goals and principles act as constraints of the solution space the framework has to explore.

This kind of domain-aware approach enables a principled development of ML solutions, where not only technical components of the solution are considered, but also domain-specific reasons and decisions behind them. Fulfilling this requirement entails multiple benefits. First, domain experts can specify and compare ML solutions based on goals and principles they are familiar with, and not just in ML-related technical terms. Second, past ML solution implementations, including their domain-specific descriptions, can serve as a knowledge base offering ML solutions for the company. Third, the documentation of principles can serve as evidence for regulatory or compliance affairs.

3.2.2. Efficient and Accurate Data Utilization

Manufacturing data are heterogeneous in many ways. Each process generates different amounts of data in varying formats (e. g., free text, images, sensor data), and with different quality levels. Yet in all cases, these data characteristics influence how a certain ML solution performs in terms of both its execution time and the quality of its results. Hence, the framework has to be aware of all diverse data characteristics to ensure an efficient and accurate data utilization. To get the necessary information about data characteristics, the framework may for instance investigate relevant data sources and metadata describing these sources. In this way, it can identify ways (potential best practices) to preprocess data in order to improve the data characteristics and thus also the effectiveness of an ML solution. Furthermore, this also helps to better reflect the original data properties in the selection and configuration of ML solutions (see [Section 3.2.4](#)).

3.2.3. Consideration of Available IT Resources

ML algorithms can be implemented in different ways, each one having positive and negative aspects regarding processing speed, robustness, or execution costs. Thus, the choice of the ML algorithm implementation affects the number of IT resources needed, i. e., computing infrastructure and software.

It is thus important that the framework takes into consideration the available IT resources and the capabilities that these resources offer. This information can be used to determine which ML solutions can be implemented with the resources available for the use case. Moreover, the documentation of the IT resources needed by different ML solutions allows their comparison at an implementation-oriented level. This means that implementation-oriented aspects, e. g., *implementation effort*, can be added to the analysis criteria in the decision-making process. Besides, it can contribute over time to a homogenization of ML solutions, allowing them to run on a common platform for all use cases in a company.

3.2.4. Property- and Performance-based Selection and Configuration of ML Algorithms

ML algorithms are at the core of an ML solution. One important step towards making their selection and configuration transparent to domain experts is establishing a base set of criteria that justify a given result. To this end, the framework must be able to select and configure an ML algorithm that is compatible with the identified domain-specific problem, data characteristics, and available IT resources, as described in Sections 3.2.1 to 3.2.3. These three aspects have to be matched with known properties of ML algorithms, e. g., the ability to produce correct and understandable rules or to handle many classification categories.

Furthermore, the framework must establish relationships between specific configurations of ML algorithms and their observed performance. Thereby, performance not only covers execution time or result quality of an ML algorithm. Instead, it also (or even mainly) has to consider whether the user's domain-specific problem has been solved by applying the algorithm. Altogether, this enables the target-oriented selection of a suitable algorithm, as well as its proper configuration for the use case at hand.

3.2.5. Enhanced ML Solution Comprehensibility

By considering all these different aspects as described so far, a framework may also risk reducing the ease for users to perceive differences among several candidate ML algorithms. Even when comparing two almost identical ML solutions, e. g., which use the same data sets and run on similar IT resources, finding a single difference responsible for a considerable performance change can quickly become error-prone and time-consuming. Hence, the framework needs to present many ML solutions simultaneously and to make their exploration and comparison as intuitive as possible. This includes explaining the user individual ML solution components in an easy-to-understand manner for domain experts and decision makers.

Thereby, the framework especially has to reflect both the goals and skills of a typical domain expert in manufacturing. Such domain experts are usually not interested in technical aspects such as parameter values of algorithms or used IT resources. Instead, they rather focus on the performance of applying an ML algorithm, measured in multiple dimensions, e. g., execution time, prediction quality, or development costs. For that purpose, the framework must (1) present ML solutions in terms of their relative performance differences, (2) offer meaningful descriptions of these performance differences, and (3) indicate patterns that most influence performance.

3.3. ML Solution Framework

The ML solution framework addresses the design requirements presented in the previous section via a structured development process, metadata profiling of solution components and an ML solution viewer to review the implemented ML solutions. The development process defines the sequence of activities to develop solution components and to create the corresponding metadata profiles. The metadata profiles define the information that needs to be documented to ensure reproducibility and improve cooperation among members of the development team. Finally, the ML solution viewer makes it easy to review and compare the developed ML solutions, thus facilitating the

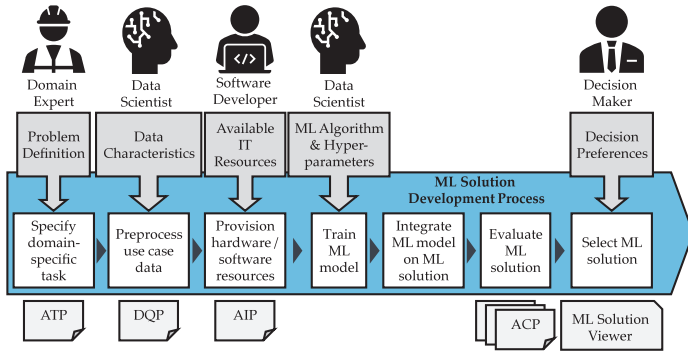


Figure 3.2.: *ML Solution Development Process*: A structured approach to create an ML solution, where experts are supported on particular topics

selection of a suitable ML solution. The following three subsections describe each part of the framework in more detail.

3.3.1. ML Solution Development Process

The development process proposes a structured approach to building ML solutions for different use cases with a common structure. An overview of the process is shown in [Figure 3.2](#). It consists of seven steps that a development team carries out not only to develop ML solution components, but also to generate the necessary metadata that is needed to understand, compare, reproduce, deploy and manage ML solutions later. The development process defines responsibilities for each of four team roles, i. e., domain expert, data scientist, software developer and decision maker. It also specifies the input that each team role has to provide (shown as gray boxes beneath the team roles) and the expected output metadata of each step (shown as gray icons beneath the process steps). The development process thus serves as a kind of *ML solution production line* that generates ML solutions for specific use cases.

Specify domain-specific task. In the first step, the domain expert (and to a lesser extent the decision maker) must provide information to define the problem that the ML solution must address. The information for the problem definition includes general constraints set by the decision maker for the development project, e. g., allocated financial budget, time frame, or manpower. With the general boundaries of the development project defined, domain experts provide domain-specific requirements and preferences regarding use case data and the different solution components. The requirements and preferences specify how the domain experts expect the use case data to be read and processed, as well as the type of results they obtain from the ML solution once available. It is important that the information collected at this step focuses on domain-specific aspects and is documented as such, i. e., that the metadata identifies this as domain-specific information, in contrast to software specifications. For this reason, the output of this step is the analytics task profile (ATP). This profile groups the metadata collected thus far.

Preprocess use case data. In the second step, the data scientist examines the data available at the use case and devises the preprocessing steps needed to retain only relevant data features and present them in a format that can be used to train a ML model. For this purpose, the data scientist consults the domain experts to determine the correct meaning and representation of the use case data based on the domain-specific semantics of each data feature. This implies, e. g., distinguishing between dates, timestamps and sensor values, all of which can be stored as float values. Prior to any preprocessing task, the data scientist must set and document common seed values in all data preprocessing libraries to be used in order to avoid inconsistencies with non-deterministic functions. This can include, e. g., data sampling or hashing functions. Each data feature is then preprocessed according to its type to make it usable for ML algorithms.

The data scientist must document the sequence of data transformations performed for each feature type, i. e., numeric, categorical, unstructured text and time data. With data features preprocessed into usable formats, the

remaining data features are sampled and split into training and test data. This ensures that the ML model is trained on data different to the one used to test its performance. The step's outputs include: the stored data splits, a specification of a data preprocessing pipeline to be used by ML models, and the step's metadata, collected with an identification code in a data quality profile (DQP).

Provision hardware and software resources. The third step is performed by the software developer, who knows which IT resources are available to the use case and can assess whether these suffice for the development. The software developer first collects the metadata about data processing from the data scientist to determine the system requirements needed. These metadata also restrict the types of ML algorithms, i. e., the compatible ML implementations, that can be applied to the resulting data set. This enables the software developer to allocate and document the computing infrastructure to handle the data volume of the use case. This should also consider the speed at which data is generated and preprocessed. The software developer can then install and document the software to handle the different data types on the allocated computing infrastructure. The step's first output is the supplied hardware and software stacks. The metadata describing these stacks is stored in the step's second output, the analytics infrastructure profile (AIP).

Train ML model. In the fourth step, the data scientist experiments with different component configurations, ML algorithms and its parameters to find good-performing ML models. This step begins an iteration loop that continues to the following two steps, as data scientist and software developer iterate over possible combinations of data features, software components and ML algorithms and techniques. The data scientist defines at least one ML algorithm, the corresponding training settings and hyperparameter values in order to train an ML model. Among the training settings is included the number of folds to use on the training data split. The strategy that he

or she uses to determine each of these elements is subject to his or her experience and the use case conditions. This is to contemplate complex ML techniques, e. g., AutoML systems, model ensembles, parameter optimizers or reinforcement learning approaches. It is important that the metadata documenting the ML model characteristics is generated, e. g., through code annotations or code profilers. These metadata is later required to produce the analytics configuration profile (ACP). This step's only output is the ML model code.

Integrate ML model into the ML solution. In the fifth step, the data scientist and the software developer integrate the trained ML model with the data preprocessing pipeline developed earlier and with any software components that trigger actions based on the model predictions. For this purpose, they transform the ML model from the previous step into a deployable software component. This can be in the form of Docker images, serializable files, e. g., pickle files in Python, or export formats, such as PMML and ONNX. Also during this step, the software developer documents any new software dependencies that results from this transformation with corresponding metadata to be included in the ACP. At this step, the ML solution is complete.

Evaluate ML solution. In the sixth step, the data scientist and the software developer evaluate the performance of the complete ML solution. For this purpose, the data scientist first defines an evaluation procedure and performance metrics that correspond to the use case requirements stated by the domain experts in the DQP. The ML solution performance is then evaluated following the evaluation procedure and the training data split created in step two. In the case that this step has already been performed as part of previous iterations, the evaluation procedure must be expanded to include comparative evaluations against the other ML solutions. This is to produce comparison data that facilitates the review of multiple ML solutions later. In all cases, the performance measured through the evaluation procedure must be documented with comprehensive metadata. The step's output groups

these metadata with that of the previous two steps to produce the analytics configuration profile (ACP).

Select ML solution. The seventh and final step is responsibility of the decision maker. Based on the metadata collected throughout the development process, he or she decides which ML solution(s) are to be deployed in the use case. For this purpose, he or she needs to review and compare the characteristics, advantages and disadvantages that each candidate ML solution entails. To facilitate these tasks, the ML solution framework provides the ML solution viewer. The step's output is then the selected ML solution.

3.3.2. Metadata Profiles

An ML solution is documented with four metadata profiles: [Analytics Task Profile \(ATP\)](#), [Data Quality Profile \(DQP\)](#), [Analytics Infrastructure Profile \(AIP\)](#), and [Analytics Configuration Profile \(ACP\)](#). Each profile documents a specific ML solution component as it is developed. An overview of the metadata profiles and the components that they describe is shown in [Figure 3.3](#). They address the design requirements discussed in [Sections 3.2.1 to 3.2.4](#). The following paragraphs explain each metadata profile in more detail.

ATP: Analytics Task Profile. This profile specifies the problem to be addressed by the ML solution, thus dealing with the requirement *domain-specific problem definition* (see [Section 3.2.1](#)). Domain experts, e. g., production planners, workers of the production line, or quality control engineers, define the task to be executed, the domain-specific goals to be achieved, as well as the conditions and principles to be observed. In this context, the ATP can only be specified using terms or concepts that are familiar to the experts. A specification process based on the well-known [QFD](#) (Akao and King, 1990) can serve this purpose. The elements covered in this profile and the way they relate to concepts in the [QFD](#) methodology can be exemplified as follows:

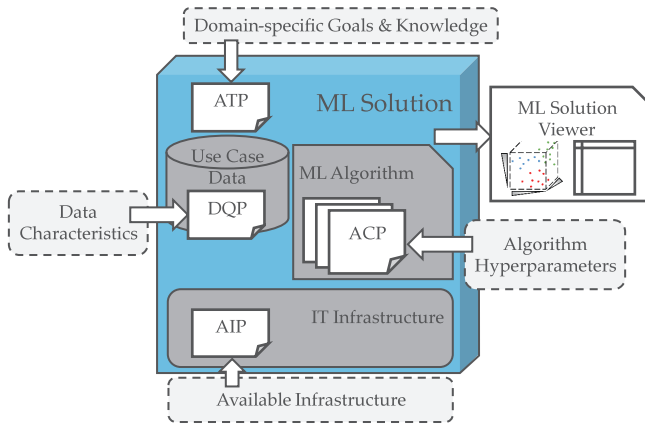


Figure 3.3.: Metadata Profiles of the ML Solution Framework

Task. This corresponds to the *product definition* for which a QFD process can be started, e. g., reduce the time it takes to diagnose defective units while performing the final quality check in the production line A6.

Goals. These express the *product functions* and *quality characteristics* (*Hows*), and their corresponding target values, e. g., decrease the time to determine the kind of defect by at least 10%.

Conditions. These state the manner in which the ML solution (product) has to fulfill its task to satisfy the domain expert’s need (*Whats*), e. g., provide more than one suggestion for the kind of defect to identify, use only the computers available for the quality check process.

Principles. They represent the trends observed in the correlations between whats and hows and the interactions between hows, e. g., for similarly fast detection methods, prefer those with higher quality over affordable ones.

Domain experts can furthermore set priorities among different goals or principles by using methods such as the Analytic Hierarchy Process (Saaty, 1987).

DQP: Data Quality Profile. This profile describes the characteristics of the available use case data as well as any data preprocessing techniques applied on it. This addresses the requirement *efficient and accurate data utilization* (see [Section 3.2.2](#)). Thereby, the **DQP** summarizes all data characteristics as analyzed by the data scientist for later related steps, i. e., model training and evaluation of the ML solution. For instance, this includes the amount of data in individual sources, different source formats and their heterogeneity, as well as the actual data quality. The **DQP** thereby describes data characteristics at the level of a whole data set (*data set summary*) and at a more detailed level, e. g., considering also individual data features. Continuing with the use case introduced earlier, the **DQP** may contain the following elements:

Data summary description Number of defective units described in the data set, number of features (variables) describing each unit, number and share of data types of the features (numerical, categories, unstructured text), file sizes, or storage type.

Individual features Data type, percentage of missing values, median or average value of the measurement, or number of outliers.

AIP: Analytics Infrastructure Profile. Concerning the requirement *consideration of available IT resources* (see [Section 3.2.3](#)), this metadata profile specifies information about the available IT resources on the computing infrastructure (CPU cores, storage space), platform (operating systems, database management systems), and software (ML libraries, data preprocessing software) levels (Mell, Grance, et al., 2011). Especially cloud-based platforms have many configuration management tools that can be used to enact this profile, such as Chef and Puppet or OpenStack's Heat¹.

Nevertheless, an **AIP** not only describes these available IT resources. In addition, this profile also specifies the capabilities of these resources with respect to handling data with different characteristics, as well as ML algorithms and their varying properties.

¹<https://www.chef.io/chef/>, <https://puppet.com>, and <https://github.com/openstack/heat> respectively

ACP: Analytics Configuration Profile. This profile completes an ML solution specification. In this context, it is important to note that, given the iteration loops in the ML solution development process (see [Section 3.3.1](#)), several candidate [ACP](#) profiles can be created for the same use case. This means that multiple ML solutions can be used to support the use case. Each of these candidate [ACP](#) profiles fulfills the fourth requirement *property-based and performance-based selection and configuration of ML algorithms* (see [Section 3.2.4](#)). The [ACP](#) includes a description of how data has to be transformed so that it may be used as input for the relevant ML algorithm (feature generation). Additionally, an [ACP](#) has to specify the methods that may be used to remove unnecessary, redundant, or low-quality data (feature selection). The most important part of an [ACP](#) is the parameterization of the selected ML algorithm itself (algorithm configuration). In the context of the use case discussed earlier, the [ACP](#) profile may contain elements like:

Feature Generation. Method to read the variables from the data set (e. g., identify failure descriptions in the worker's report: "pressure too low in injector VRM-001").

Feature Selection. The technique to filter out variables that do not help distinguish the defect type (e. g., correlation-based, principal component analysis).

Algorithm Configuration. The ML algorithm to use and the values to use for its parameters (e. g., decision trees with a Gini impurity of 0.42). Furthermore, the [ACP](#) stores results of evaluating predefined performance metrics. These performance metrics are chosen based on their ability to contribute to the achievement of domain-specific goals defined in the [ATP](#). Thereby, the [ACP](#) also documents the evaluation procedure in which these measurements were obtained. This information enables the reproducibility of performance results.

3.3.3. ML Solution Viewer

In the end, decision makers compare ML solutions using the *ML Solution Viewer*, which is a dashboard with easy-to-understand information for ML

non-experts. The viewer facilitates the comparison of candidate ML solutions, and thus the final selection of the most suitable solution. It hence addresses the requirement *enhanced ML solution comprehensibility* (see [Section 3.2.5](#)).

The ML Solution Viewer processes metadata from all generated profiles. It then presents domain-oriented visualizations and information to decision makers so that they can explore and understand ML solutions intuitively. This does not mean that the ML solution viewer only serves a showcasing purpose. It also analyzes how decision makers use the interface in order to identify patterns in their usage behaviors. This way, the viewer can learn the users' preferences and adapt its visualizations to be more useful.

Similarly, the ML solution viewer can retrieve and provide ML experts with the original technical specification of each of the displayed ML solutions. This is done to provide additional transparency and to open the possibility to further tune the ML solution beyond the options considered by the framework.

3.4. Prototypical Implementation and Discussion

This framework was developed into a prototype using the R and Python programming languages, as well as JSON as the file format to write the profiles (see [Section 3.3](#)). Implementations of ML algorithms are provided by the WEKA data mining software ¹ and the python library scikit-learn ².

The ML solution viewer was implemented as a web client composed of three modules. The prototype's module architecture is shown in [Figure 3.4](#). The first module enables the web interface and was implemented using the python library Dash ³. The visualizations shown in the web client were implemented using plotly ⁴. The third module parses the data contained

¹<https://www.cs.waikato.ac.nz/ml/weka/>

²<https://scikit-learn.org/stable/>

³<https://plotly.com/dash/>

⁴<https://plotly.com/python/>

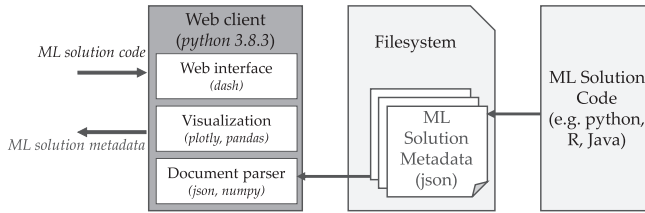


Figure 3.4.: System Architecture of the ML Solution Viewer

in JSON files to populate the web interface and visualizations. It uses the standard python module `json`¹, and the libraries `numpy`² and `pandas`³.

As ML solutions are evaluated in the ML solution development process, the executed solution code stores the profile metadata as JSON files at a predetermined directory. An example of the metadata stored in these JSON files is provided in [Appendix A](#). Upon its deployment, the document parser module of the ML solution viewer parses all available JSON files in the directory mentioned above. As a result, the ML solution viewer can be accessed through a web browser (see [Figure 3.5](#)).

The viewer offers three drop-down controls on the left side to access the metadata for a specific ML solution. The first drop-down lists the ML algorithm implementations documented in the profiles. The implementation also includes solutions based on the following algorithms: decision trees (DTR), naive bayes (NBY), random forests (RFR), k-nearest neighbors classifier (KNN), deep learning (DLN), and logistic regression (LGR). The second drop-down lists the acronyms of all the use cases for which ML solutions have been developed. The third drop-down lists the numeric identifiers of the solutions developed for that use case and ML algorithm combination. The concatenation of the three values constitutes the identification code of an ML solution and all its corresponding metadata profiles. In the example shown in [Figure 3.5](#), the values of the drop-down lists point to the solution

¹python libraries

²<https://numpy.org>

³<https://pandas.pydata.org>

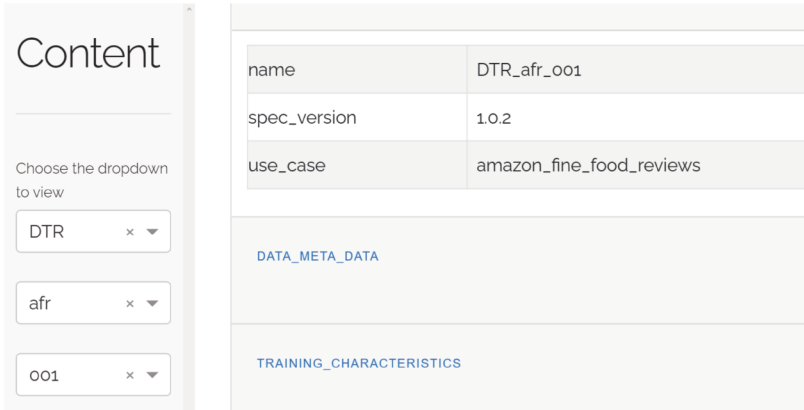


Figure 3.5.: Web prototype of the ML Solution Viewer

with the code "DTR-afr-001". This is the first solution implemented with a decision trees algorithm for the use case "afr" (afr stands for *amazon fine food reviews*).

Once an ML solution code is formed, the document parser module retrieves the complete metadata for that solution and summarizes it in thematic tables and visualizations. This is to avoid the information overload that would represent displaying the JSON files in their entirety. For instance, the performance metrics from the [ACP](#) profile are displayed using intuitive visualizations, tables or summary values, as shown in [Figure 3.6](#). Moreover, the viewer allows users to blend in or blend out the information displayed, so that they can focus on particular aspects of the ML solution.

The prototype was used with public data sets from different application areas and containing different data types. This allows a more robust assessment of the prototype's capabilities. The use cases for this implementation were all classification tasks, regardless of the feature types they had, i. e., numeric, categorical, unstructured text data or date features.

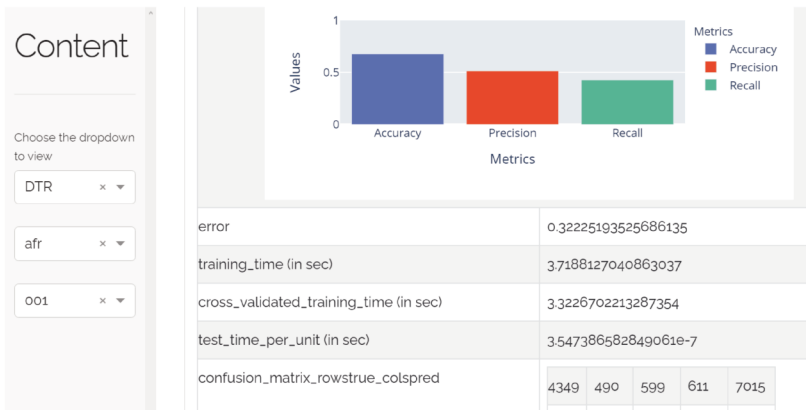


Figure 3.6.: ML solution viewer displaying details of performance metadata

All use cases measured their performance in terms of execution costs, execution time, and quality (a combination of accuracy, precision and recall).

The use of the ML solution framework helps reduce the number of iterations needed to develop suitable ML solutions. Moreover, the development work becomes more structured and robust, as the development process ensures that reproducibility data is generated and that the necessary activities are performed every time.

Aside from these test use cases, other potential use cases in manufacturing are compatible with this approach. This is possible if they share a similar underlying classification task. In this context, the goal is to help domain experts to identify a particular category, e. g., of a root cause, error type, or the most suitable solution to a problem (Langley and Simon, 1995; Pham and Afify, 2005; Wuest et al., 2016). Such use cases exist in any phase of the product lifecycle, e. g., in production quality control, predictive maintenance, or after sales customer support.

3.5. Summary and Future Work

This chapter introduced an ML solution framework to systematically develop ML solutions. This framework reflects relevant design requirements via a common development process, four metadata profile types and an ML solution viewer. For a concrete use case, the ML solution framework ensures that the development of the ML solution truly responds to the domain-specific requirements set by domain experts. It also devises a comprehensive development process to develop all solution components, with the collaboration of the relevant stakeholders and in observation of the necessary reproducibility practices. Finally, the ML solution viewer provides easy-to-use means to visually compare the developed ML solutions, making it easier for decision makers to make an informed selection.

The framework was implemented in an experimental prototype and evaluated against publicly available data sets. Thus, the prototype can validate the framework's feasibility and usefulness. In particular, it results in a systematic approach that significantly helps domain experts from manufacturing to configure and select an ML solution that best matches their requirements. It also serves as the basis for the remaining contributions. Specifically, the framework establishes the need for a dedicated design step to ensure that domain-specific needs truly steer ML solution development (see [Chapter 4](#)). Additionally, the ML solution framework generates metadata that can be consolidated in an ML solution repository to recommend and reuse ML solutions (see [Chapter 5](#)).

Future research includes the extension and evaluation of the framework to cover other kinds of use cases, e. g., for scrap reduction. This may also require adding support for further data formats or ML algorithms, as needed by the respective use cases.

CHAPTER



AXIOMATIC DESIGN FOR MACHINE LEARNING

As ML solutions grow in size and complexity (Sharp et al., 2018), an ML solution *specification* becomes necessary to prescribe how to implement the ML solution. This specification includes a description of the required data input and output and the functionality that the solution shall provide. The ML solution specification also describes software requirements, as well as internal settings, e. g., the hyperparameters to tune the ML algorithms.

ML solution specifications are the result of a design process. Yet, this design process is not straight-forward (see the *design challenge* CH-1 in [Section 1.3](#)). Concretely, it faces three feasibility requirements: (1) the creation of a specification through systematic experimental learning; (2) the clear documentation of the design intention behind this specification; (3) the end-to-end traceability and consistency of design choices. Existing related approaches do not fully address these requirements (see [Section 4.7](#)), thus rendering the design process more complex and its outcome less predictable.

Axiomatic Design is a methodology to systematically design complex products (Suh, 2001). This chapter proposes several adaptations to Axiomatic Design that enable the design process to handle the feasibility requirements. This adapted design methodology is called AD4ML. This chapter shows the use of AD4ML to specify an ML solution for a fault detection use case and discuss the improvements that the approach brings to the design process. These improvements include the visualization of ML solution specifications, the agile design of ML solutions, and the reutilization of specification components in different use cases. Thereby, the chapter demonstrates how AD4ML tackles the feasibility requirements. The chapter also shows how AD4ML can be leveraged to early validate and assess ML solution specifications. This allows the identification of design flaws already while the specifications are being created.

The contents of this chapter are an edited version of separate publications (Villanueva Zacarias et al., 2020; Villanueva Zacarias, Ghabri, et al., 2021). All concepts extracted from them and used in this dissertation are the original work of the dissertation author.

This chapter is organized as follows: Section 4.1 introduces a fault detection use case and details the three *feasibility requirements*. Section 4.2 gives an overview of Axiomatic Design. Section 4.3 discusses the adaptations to Axiomatic Design that constitute AD4ML and the improvements that AD4ML brings to the design process. Section 4.4 outlines how AD4ML can be leveraged to validate and assess ML solution specifications. Section 4.5 describes a prototypical implementation of AD4ML called the *ML solution designer*. Section 4.6 discusses and assesses how AD4ML addresses the three *feasibility requirements*. Section 4.7 discusses the related work whereas Section 4.8 concludes.

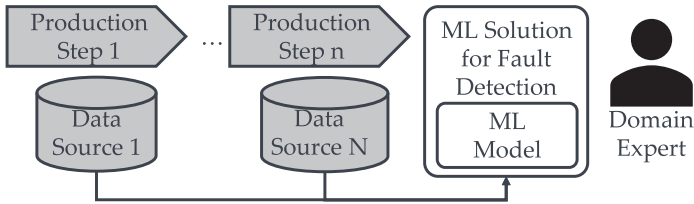


Figure 4.1.: Overview of the example use case for *Fault Detection*

4.1. Problem Context and Requirements

This section first describes a common use case of supervised learning in manufacturing, and then discuss the feasibility requirements for designing ML solution specifications.

4.1.1. Use Case: Fault Detection in a Production Line

Figure 4.1 shows the use case taking place in the final quality check at the end of a production line. In this production line, different types of nozzles are produced. At every step, measurement data for each nozzle is collected. These measurements can be numeric, e. g., a *flow rate*, or categorical, e. g., *leak test approved*. At the end of the production line, an inspector reviews the measurement data to decide if the nozzle is functional or defective. This is commonly called *fault detection* (Isermann, 2017).

Manually reviewing the data leads to a big overhead for the inspector. Instead, an ML solution can be designed and implemented to predict the defects automatically. This corresponds to a binary classification task. The ML solution can read and process all measurement data to identify which measurements and which of their values correlate with the presence of defects. A supervised learning algorithm can be used to train a machine learning model that predicts the existence of defects based on these correlation patterns. In the end, the ML solution can report the measurements that triggered the decision for each potentially defective nozzle.

4.1.2. Feasibility Requirements in the Design of ML Solutions

Figure 4.2 depicts a sequence of three milestones in the design process of an ML solution. For the first milestone, domain experts state their requirements by defining a *design intention*. This is a domain-specific description of the functionality that an ideal solution should have, and the conditions in which the solution should operate. These conditions include the use case needs, goals, assumptions, operational or budget constraints, and performance priorities. For the second milestone, a multidisciplinary team creates a *ML solution specification*. This is different from the design intention in that it describes a working and viable implementation of the ML solution. The third milestone is the actual implementation of the selected ML solution specification. This design process faces three concrete requirements (shown in gray in Figure 4.2). These feasibility requirements are additional to the well-known requirements of training ML models, e. g., collecting enough high-quality data that are statistically relevant.

R₁. Enablement of systematic experimental learning. This requirement affects the creation of the *ML solution specification*, i. e., the second milestone of the design process (see Figure 4.2). Traditional software systems are based on clear information requirements and technology specifications that are available upfront (Marchand and Peppard, 2013). ML solutions, on the contrary, need extensive experimentation in their design and development (Viaene and Van den Bunder, 2011). A ML solution is the combination of three types of components (see white boxes in Figure 4.2). The first type are software components for processing input or output data. The second type is the ML algorithm used to train a model with certain hyperparameters. The third type is the computing infrastructure needed to deploy the ML solution. There are many possible combinations of these components. Identifying which combinations of components fulfill the design intention needs experimentation. Experimentation is however costly and time-consuming, as it involves implementing and evaluating different prototypical solutions.

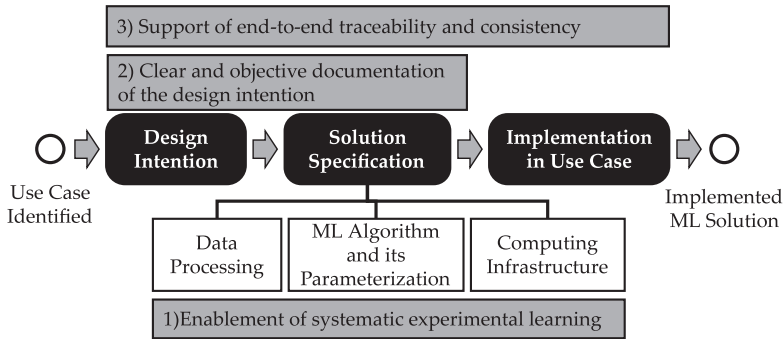


Figure 4.2.: Overview of the milestones (in black) and requirements (in gray) in the design process of Machine Learning Solutions

Without a plan or structure, the amount of experimentation required to design a viable specification becomes burdensome.

The team designing the ML solution specification must follow a systematic process that allows them to learn from experimentation. For this purpose, the design process must support the comprehensive documentation of experiments with different ML solution components. Having this documentation makes it easy to decide how to change, extend or reduce the components in the specification. For such modifications to be practicable, the specification must also provide an easy way to identify the impact of those modifications on the rest of the specification components.

R₂. Clear and objective documentation of the design intention. This requirement highlights the need for a documented design intention. An ML solution specification can only describe a concrete implementation of the required functionality. However, this is no substitute for the design intention. The latter describes required functionality in a use case context. Maintaining a complete description of a design intention offers a clear and objective reference for several purposes. First, it makes the intention of an ML solution explicit. This eliminates the subjectivity when interpreting the use of a

software component. Second, it allows domain experts to understand an ML solution based on its usefulness for the use case. Third, it provides an objective evaluation basis to select suitable ML solution specifications.

R₃. Support of end-to-end traceability and consistency. This requirement refers to the need of knowing the *end-to-end* line of thought that turns a design intention into an implemented ML solution. Experimenting with different component combinations produces many ML solution specifications. However, none of them is guaranteed to completely fulfill the given design intention. There are cases where compromises need to be made to obtain the desired functionality. Such compromises have to be satisfactory for everyone in the multidisciplinary team of data scientists, software developers, and domain experts.

There needs to be a trace of the decisions that lead to a complete ML solution specification. Having the line of thought documented from end-to-end of the design process offers multiple benefits. First, it is easier to verify which functionalities have been fulfilled and which have been left out. Second, it is possible to identify at which point of the design process compromises were made. Third, stakeholders can check if their decisions are consistent with one another, and with the decisions of their colleagues.

4.2. Main Concepts of Axiomatic Design

Axiomatic Design (AD) is a methodology that aims to systematically generalize the design process of complex products (Rauch et al., 2016; Suh, 2001), such as domestic appliances, industrial machinery, and manufacturing systems. This section gives a summary of the main concepts of this methodology in their original context.

To generate a product design, four different roles engage in the design process as shown in [Figure 4.3](#). Each role creates a certain kind of design element to contribute to the product specification. A product owner in the *customer domain* formulates *desired Customer Attributes (CAs)* to outline the

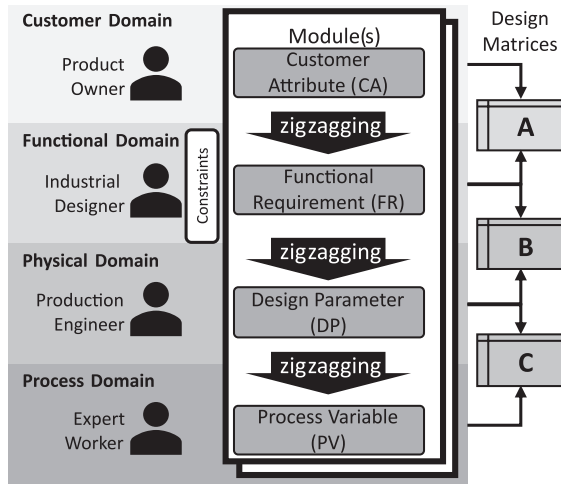


Figure 4.3.: Conceptual overview of Axiomatic Design along with the roles involved in their definition.

scope of the product. These customer attributes can be physical properties of the product like *withstanding certain temperatures without deformation*. Afterwards, industrial designers in the *functional domain* match these CAs to *Fehlerrates (FRs)*. The functional requirements can be product functionality to achieve the attribute like *thermal dissipation*. Production engineers in the *physical domain* convert FRs to *Design Parameters (DPs)* that are matched to *Process Variables (PVs)* by domain experts in the *process domain*. The design parameters can be product parts that enable the functionality, like *layers of different materials*. Process variables can be key values that characterize well-made product parts, like *thickness of the material layers*. All design elements matched from the same initial CA constitute a *module* (see black-framed grouping box in Figure 4.3). Besides design elements, a product specification can also have input *constraints*. They set boundary conditions applicable to the overall design (Suh, 2001), e. g., maximum production costs. Constraints are set on the functional domain and apply for this and other domains afterwards.

Design elements also need to be *decomposed* into more concrete elements. For that decomposition, the design elements of neighboring roles have to be considered. The design process and the product specification are finished when all kinds of design elements have been decomposed in sufficient detail so that the desired product may be manufactured. This process of matching and decomposing design elements is called *zigzagging*. It goes over the four kinds of design elements: CAs in the *Customer Domain*, FRs in the *Functional Domain*, DPs in the *Physical Domain* and PVs in the *Process Domain* (Suh, 2001). The specifics of the zigzagging process are explained later in Section 4.3.

Matches between design elements are represented in design matrices, as shown in Figure 4.3. In each matrix, neighboring design elements form the rows and columns of the matrix. Matched elements are given a coefficient different than zero in their intersection. With this information, relationships between design elements can be expressed as mathematical equations. Axiomatic Design calls them design equations. For instance, assume a matrix A that relates two CAs (rows) and two FRs (columns). This is shown in equation form in Equation (4.1).

$$\begin{Bmatrix} CA_1 \\ CA_2 \end{Bmatrix} = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \begin{Bmatrix} FR_1 \\ FR_2 \end{Bmatrix} \quad (4.1)$$

Two design equations describe the relationships in this matrix

$$CA_1 = A_{11}FR_1 + A_{12}FR_2 \quad (4.2)$$

$$CA_2 = A_{21}FR_1 + A_{22}FR_2 \quad (4.3)$$

where A_{ij} are the coefficients in row i and column j of design matrix A. In general, a design equation to describe a CA is:

$$CA_i = \sum_{j=1}^n A_{ij}FR_j \quad (4.4)$$

where n is the number of FRs. This can also be written as

$$\vec{CA} = A \times \vec{FR} \quad (4.5)$$

where \vec{CA} and \vec{FR} are vectors of CAs and FRs. Similar equations exist for FRs in design matrix B and DPs in design matrix C.

In an *ideal design* specification, there should only be *one-to-one* matches in the design matrices. In other words, each design matrix should be a diagonal matrix. This simplifies the design equations, hence reducing the complexity of the product specification. Matrices with *one-to-many* matches are *redundant designs*. If the *one-to-many* matches can be arranged in a triangular matrix, the specification has a *decoupled design*. Decoupled designs require the product specification to be implemented in a specific sequence to work. If the design matrix is non-triangular, non-diagonal, or full, the specification is a *coupled design*. This term denotes a specification where neighboring design elements have *many-to-many* matches between them, making the design very complex.

Axiomatic Design relies on two axioms to prescribe *ideal designs*. Firstly, the *independence axiom* states that design elements should be satisfied without affecting other design elements of the same domain. So for example, the FRs satisfying one CA must be independent from other FRs and not satisfy other CAs. This excludes the possibility of producing coupled designs. Exceptions of this axiom are constraints (Suh, 2001), e. g., a constraint on the total weight of a product. If many design specifications fulfill the first axiom, the *information axiom* states that the design with the smallest information content is the best. This refers to the design with the least number of design elements that still satisfies all desired customer attributes. Formal definitions of these axioms and more details on Axiomatic Design are described by Suh, 2001.

Table 4.1.: Comparison of Axiomatic Design and AD4ML

Axiomatic Design	AD4ML
Domains	Design Layers
Customer Domain	Domain-specific Design
Customer Attribute (CA)	Domain Request (DR)
Functional Domain	Analytics Design
Functional Requirement (FR)	Analytical Concept (AC)
Physical Domain	Technology-specific Design
Design Parameter (DP)	Technological Resource (TR)
Process Domain	–
Constraint	Constraint
Module	Component
–	Type of design element
Design Matrix	Design Matrix

4.3. Axiomatic Design for Machine Learning (AD4ML)

The section first discusses the proposed adaptations of Axiomatic Design for ML solutions. Afterwards, the design of a specification for the fault detection use case is demonstrated. Thereby, the section explains the design elements that are part of this specification. This specification exemplifies the details of the zigzagging process to come up with an implementable ML solution. This section closes by discussing three improvements to the design process. The first improvement is the visualization of AD4ML specifications. The second improvement is an approach to reutilize specification components. The third improvement is an approach for the agile design of ML solutions with AD4ML.

4.3.1. Adaptations to Axiomatic Design for ML Solutions

The design of ML solutions requires several adaptations to Axiomatic Design (AD). Table 4.1 compares the concepts of AD with their AD4ML equivalents. Figure 4.4 gives an overview of the resulting AD4ML concepts.

The first adaptation consists in adapting the terminology of AD concepts to increase familiarity for domain experts, data scientists, and other software

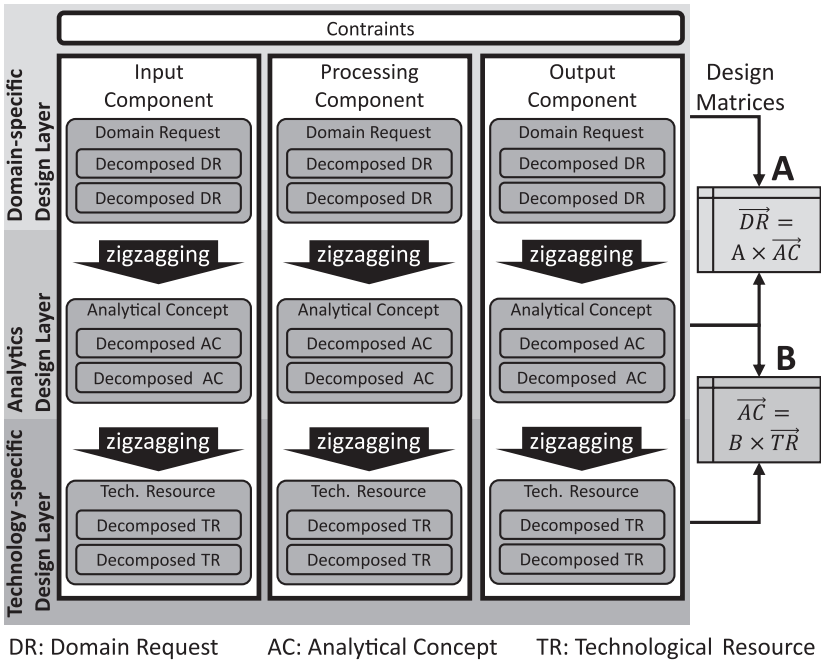


Figure 4.4.: Overview of AD4ML concepts.

developers. In AD4ML, *domains* become *design layers* based on the concept of layers in software frameworks. AD4ML design layers and design elements are also renamed based on the stakeholders they address. The *customer domain* becomes the *Domain-specific design layer*. Domain experts formulate **Domain Requests (DRs)** instead of *Customer Attributes*. DRs describe domain-specific needs to be fulfilled by the ML solution. The *functional domain* becomes the *Analytics design layer*. Data scientists specify **Analytical Concepts (ACs)** instead of *Functional Requirements*. ACs describe the data analytics functionality needed to satisfy DRs. This includes methods for loading and handling data, data set descriptions, or data analytics capabilities as defined by Gröger (Gröger, 2018), i. e., descriptive, diagnostic, predictive and prescriptive. Together, they constitute a conceptual architecture of the

data analytics functionality. The *physical domain* becomes the *Technology-specific design layer*. Software developers define **Technological Resources (TRs)** instead of *Design Parameters*. TRs describe concrete software libraries, execution platforms or technology stacks that materialize ACs. Depending on the level of decomposition, TRs can include system and version requirements, or function names and arguments. Together, they constitute a technical software specification of the ML solution.

The *Process Domain* in AD focuses on monitoring the production process of the created design (Suh, 2001). It could thus be mapped into a *deployment* design layer to monitor the operation of the implemented ML solution. Nevertheless, the focus of this chapter is on proposing an approach to specify the design of ML solutions before they can be deployed. Therefore, this chapter considers such a layer out of scope.

Constraints are set in the functional domain in AD to impose general restrictions on the specification. In manufacturing use cases, this is a task of domain experts. AD4ML thus relocates constraints in the *domain-specific* design layer. For example, domain experts may set maximum total development costs for an ML solution.

The *module* of AD is called *component* in AD4ML in correspondence to actual software components. One component comprises all design elements matched from the same initial Domain Request (DR).

Unlike Axiomatic Design, AD4ML assigns components one of three types based on the data-driven nature of ML solution solutions. All ML solutions are composed of *input*, *processing*, and *output* software components. *Input* components load, format, and preprocess data as preparation for processing components (e. g., feature extraction). *Processing* components then apply algorithms to learn patterns from the data they receive and to build a machine learning model, e. g., a classifier or a regression model. *Output* components generate results or trigger actions based on the prediction made by the machine learning model. AD4ML reflects this composition of ML solutions with the types: *input* (*i*), *processing* (*p*) and *output* (*o*). ML solution specifications in AD4ML have at least one component of each type.

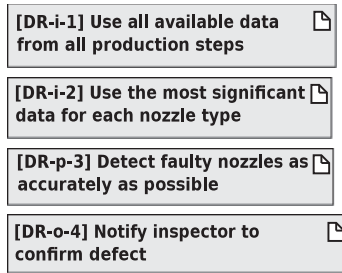


Figure 4.5.: List of initial Domain Requests for the use case example.

Matches between design elements are represented in design matrices A and B (see Figure 4.4). Matrix A captures matches between *Domain Requests* and *Analytical Concepts*. Matrix B does the same between *Analytical Concepts* and *Technological Resources*. The independence and information axioms from Axiomatic Design continue to apply to ML solution specifications created with AD4ML.

4.3.2. AD4ML Specification for the Fault Detection Use Case

As mentioned above, the design of an ML solution specification begins by defining initial DRs in the *domain-specific* design layer. Figure 4.5 shows four DRs for the fault detection use case. Each one is given a code for easy reference. The code states the kind of design element (DR), its type (i for input) and a numeric identifier.

These initial DRs specify abstract functionality attributes that domain experts desire. They have a strong domain-specific context. They are formulated as endings to the phrase "The ML solution is requested to...". For example, in the use case, DR-i-1 and DR-i-2 detail how domain experts expect data to be loaded, i. e., they want the ML solution to use all data that is available at involved production steps (DR-i-1). Additionally, they request to *use the most significant data for each nozzle type* (DR-i-2). DR-p-3 describes the ML solution's analytics task, i. e., which kind of predictions domain experts require from the solution to support their use case. In the

example, they want an accurate detection of faulty nozzles at the end of the production line. DR-o-4 may express desired attributes of the data output. Here, domain experts want the ML solution to notify an inspector about the predictions of defective nozzles.

These four initial DRs set the scope for the whole specification. This means that the example ML solution specification will finally consist of four components, each one representing one initial DR. Nevertheless, these components need to be detailed to arrive at an implementable ML solution specification. This is done through zigzagging processes of matching and decomposition. Each initial DR triggers zigzagging processes to create initial ACs and TRs via matching, as well as to detail all DRs, ACs, and TRs via decomposition. An ML solution specification is complete when all DRs, ACs and TRs have been sufficiently decomposed so that their corresponding software components can be implemented. To illustrate this idea, [Figure 4.6](#) shows the details of the zigzagging processes for DR-i-2. Zigzagging for the other three initial DRs works in a similar way.

Matching always occurs *across design layers*, following a top-to-bottom sequence in [Figure 4.4](#). The logic to match any two design elements is that elements from the upper design layer *state demands* whereas elements in the lower design layer *provide means* to fulfill them. In the example, a zigzagging process hence begins by matching DR-i-2 to an Analytical Concept (see step 1 in [Figure 4.6](#)). Such an AC describes data analytics functionality on a conceptual level. It represents a technology-independent proposal of data scientists to fulfill the request of a domain expert. For DR-i-2, a data preprocessing pipeline can be suggested. Here, the data scientist specifies in AC-i-2 at an abstract level that such a pipeline should determine which data is significant for each nozzle type. The details about individual pipeline steps that assess and filter the data appropriately are specified in later decomposition and matching steps.

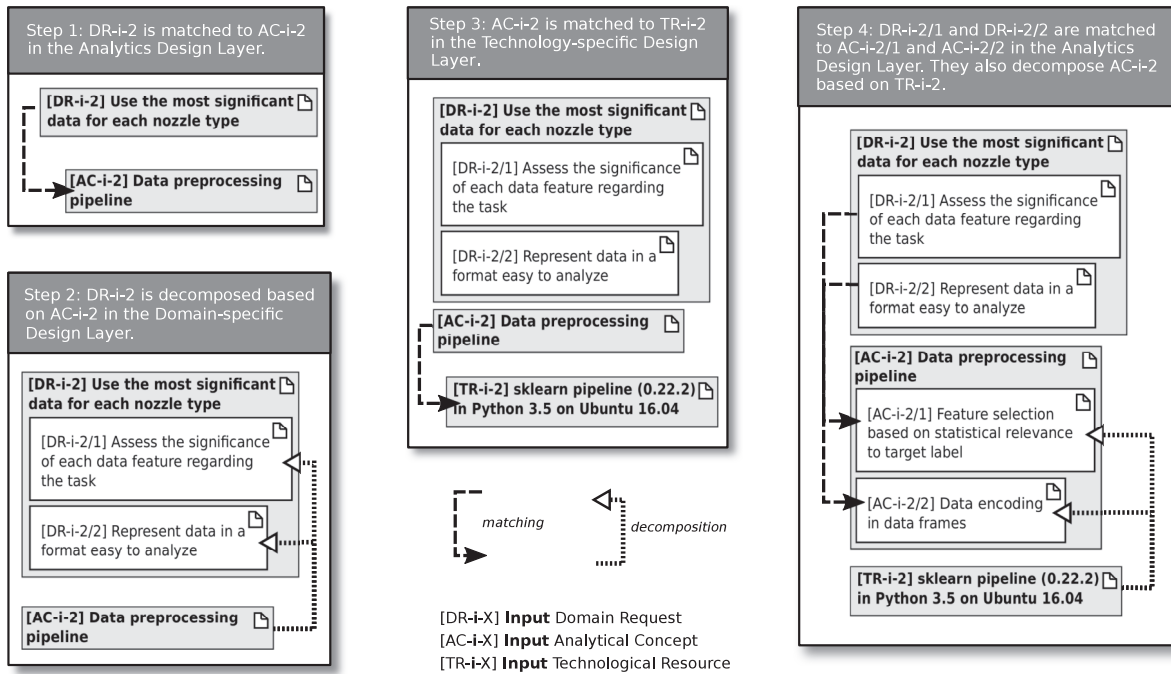


Figure 4.6.: Example of the zigzagging processes to specify details of the initial and abstract DR-i-2. *Matching arrows* are dashed and black-headed. *Decomposition arrows* are dotted and white-headed. Decomposed design elements are shown as white boxes.

This match between DR-i-2 and AC-i-2 is reflected in the *design matrix* A. Provided that other DRs also have one AC matched to them, the *design matrix* A can be expressed as:

$$\begin{Bmatrix} DRi_1 \\ DRi_2 \\ DRp_3 \\ DRo_4 \end{Bmatrix} = \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\ A_{2,1} & A_{2,2} & A_{2,3} & A_{2,4} \\ A_{3,1} & A_{3,2} & A_{3,3} & A_{3,4} \\ A_{4,1} & A_{4,2} & A_{4,3} & A_{4,4} \end{bmatrix} \begin{Bmatrix} ACi_1 \\ ACi_2 \\ ACp_3 \\ ACo_4 \end{Bmatrix} \quad (4.6)$$

The design equation for DR-i-2 thus is:

$$DRi_2 = A_{21}ACi_1 + A_{22}ACi_2 + A_{23}ACp_3 + A_{24}ACo_4 \quad (4.7)$$

where A_{jk} are the coefficients of row j and column k in *design matrix* A. Note that, different to Equation (4.4), this design equation uses indices j and k to avoid confusion with the newly introduced type i for *input* components. In general, the equation that represents *design matrix* A is:

$$DR_j = \sum_{k=1}^n A_{jk}AC_k \quad (4.8)$$

where n is the total number of ACs.

This equation includes coefficients that match the data preprocessing pipeline (AC-i-2) with all initial DRs. However, the data scientist proposed the use of a data pipeline to only satisfy DR-i-2. This means that coefficients A_{21} , A_{23} and A_{24} are zero. Only A_{22} has a non-zero coefficient. In other words, DR-i-2 and AC-i-2 have a one-to-one relationship. This fulfills the *independence axiom*. The design equation for DR-i-2 thus becomes:

$$DRi_2 = A_{22}ACi_2 \quad (4.9)$$

After this matching step, the rather abstract DR-i-2 has to be decomposed into more specific DRs (DR-i-2/1 and DR-i-2/2). Note the use of a slash in the design element codes to distinguish the numeric identifiers of each

decomposition level. This indicates they are decompositions of DR-i-2. It also prevents confusion with the potential initial DRs DR-i-21 and DR-i-22. Numeric identifiers count all design elements in a given decomposition level within the same sequence. This means, e. g., that all DRs in [Figure 4.5](#), regardless of their type, belong to the same numbering sequence. The decomposition shown in step 2 of [Figure 4.6](#) must take into account the previously matched design element AC-i-2. This is indicated by the *decomposition* arrow. This means, the details of DR-i-2/1 and DR-i-2/2 have to be consistent with AC-i-2. This is the case if they later match to decomposed ACs of AC-i-2 and they yield the complete functionality of AC-i-2. First, the pipeline must be able to assess the significance of each data feature. This requires establishing metrics to do the assessment. Second, the pipeline must also be able to manipulate data easily. This requires using a data structure that supports the assessment of significance and any further operations of the preprocessing pipeline. Together, they fulfill the need of using the most significant data for each nozzle type (DR-i-2).

To continue the design process, a new zigzagging process begins with AC-i-2. Step 3 of [Figure 4.6](#) shows AC-i-2 being matched to a corresponding Technological Resource (TR-i-2). In this case, matching takes place between the *analytics* and *technology-specific* design layers. The goal is to materialize the analytics functionality defined by *Analytical Concepts* with concrete software libraries or technology stacks that fit the company's IT landscape. In the case of matching AC-i-2, the resulting TR-i-2 has to be software capable of implementing data pipelines. A common choice for this is the Python package scikit-learn. It has multiple data transformation functions and a pipeline class to bind them together. Thus, TR-i-2 specifies scikit pipelines in an Ubuntu host to materialize AC-i-2.

The match between AC-i-2 and TR-i-2 belongs to *design matrix* B. Just like in *design matrix* A, it is assumed that each AC is matched to a single TR. This yields a *design matrix* with a form like that in [Equation \(4.6\)](#). From that *design matrix*, the design equation for AC-i-2 can be obtained:

$$ACi_2 = B_{21}TRi_1 + B_{22}TRi_2 + B_{23}TRp_3 + B_{24}TRo_4 \quad (4.10)$$

where B_{kl} are the coefficients in *design matrix* B. In general, the equation to represent *design matrix* B is:

$$AC_k = \sum_{l=1}^m B_{kl}TR_l \quad (4.11)$$

where m is the total number of TRs. Since the scikit pipeline is specified only for the materialization of AC-i-2, coefficients B_{21} , B_{23} and B_{24} are zero. The only non-zero coefficient, B_{22} , remains in the design equation:

$$ACi_2 = B_{22}TRi_2 \quad (4.12)$$

Step 4 on [Figure 4.6](#) shows the decomposition of AC-i-2 into more specific ACs (AC-i-2/1 and AC-i-2/2). However, these decomposed ACs are at the same time matches of the previously decomposed DRs, i. e., DR-i-2/1 and DR-i-2/2. Because of this, step 4 shows *matching* and *decomposition* arrows going into AC-i-2/1 and AC-i-2/2. This indicates the need for coordination between stakeholders and across all design layers. This requires to engage in a *concurrent engineering* design process (Suh, 2001). To fulfill the request to assess significance of data (DR-i-2/1), a data scientist can propose to perform feature selection based on statistical tests (AC-i-2/1). Similarly, the same data scientist can suggest to use data frames (AC-i-2/2) to facilitate analysis (DR-i-2/2). These two ACs are at the same time consistent with the matching TR-i-2 provided by the developer. This is because they can be materialized by functions of scikit-learn or other Python packages compatible with it.

The two new matches further detail *design matrix* A. This affects the row of DRi_2 and the column of ACi_2 in [Equation \(4.6\)](#). They are replaced by the

new decomposed elements DR-i-2/1, DR-i-2/2, AC-i-2/1 and AC-i-2/2. As a result, the following matrix results (new design elements in bold):

$$\begin{pmatrix} DRi_1 \\ \mathbf{DRi}_{2/1} \\ \mathbf{DRi}_{2/2} \\ DRp_3 \\ DRo_4 \end{pmatrix} = \begin{bmatrix} A_{1,1} & \mathbf{A}_{1,2/1} & \mathbf{A}_{1,2/2} & A_{1,3} & A_{1,4} \\ \mathbf{A}_{2/1,1} & \mathbf{A}_{2/1,2/1} & \mathbf{A}_{2/1,2/2} & \mathbf{A}_{2/1,3} & \mathbf{A}_{2/1,4} \\ \mathbf{A}_{2/2,1} & \mathbf{A}_{2/2,2/1} & \mathbf{A}_{2/2,2/2} & \mathbf{A}_{2/2,3} & \mathbf{A}_{2/2,4} \\ A_{3,1} & \mathbf{A}_{3,2/1} & \mathbf{A}_{3,2/2} & A_{3,3} & A_{3,4} \\ A_{4,1} & \mathbf{A}_{4,2/1} & \mathbf{A}_{4,2/2} & A_{4,3} & A_{4,4} \end{bmatrix} \begin{pmatrix} ACi_1 \\ \mathbf{ACi}_{2/1} \\ \mathbf{ACi}_{2/2} \\ ACp_3 \\ ACo_4 \end{pmatrix} \quad (4.13)$$

It is known that the data scientist proposed exactly one AC for one DR. This simplifies the design equations for DR-i-2/1 and DR-i-2/2. Most coefficients in each design equation become zero, except for $A_{2/1,2/1}$ and $A_{2/2,2/2}$, which are at the diagonal of the matrix:

$$DRi_{2/1} = A_{2/1,2/1}ACi_{2/1} \quad (4.14)$$

$$DRi_{2/2} = A_{2/2,2/2}ACi_{2/2} \quad (4.15)$$

At this point, the component specification for DR-i-2 contains 7 design elements. Beyond step 4, the zigzagging processes continue to decompose all design elements similarly until all solution components can be implemented. This occurs when design elements describe fully-configured, readily-available software. For AC-i-2/1, this is the case when the scikit-learn functions and the threshold values to determine statistical relevance are specified.

4.3.3. Visualization of ML Solution Specifications

ML solution specifications may become very complex after several decomposition steps. It is thus useful to have a visual representation of the complete ML solution specification. AD4ML uses *design matrices* A and B to create a flow diagram for the specification. For this purpose, both matrices must have consistent matching and decomposition relationships. The resulting flow diagram shows the specification components and the relationships among

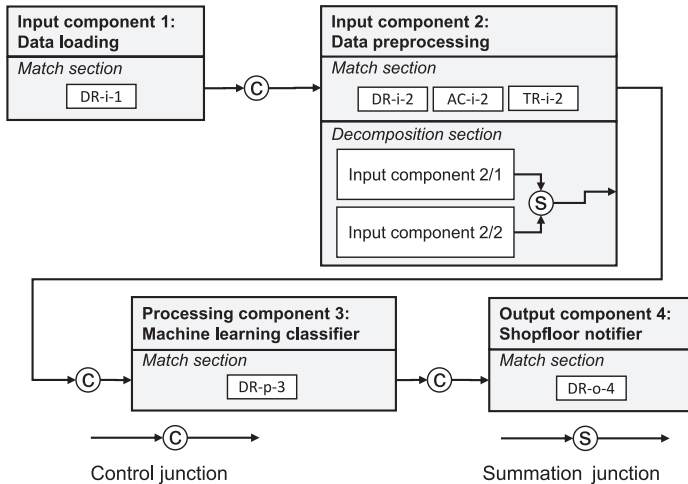


Figure 4.7.: Flow diagram of the ML solution specification introduction in Section 4.3.2

them. The flow diagram from Axiomatic Design (Suh, 2001) is extended to support the design process. This includes depicting every specification component as a form with a header and up to two sections. The first section displays *matched* design elements, while the second section displays *decomposed* components.

After the four steps in Figure 4.6, the *design matrix* A is described by Equation (4.16). *Design matrix* B has consistent non-zero coefficients, because AC-i-2 is matched only to TR-i-2, as shown in Equation (4.12). This means that both matrices are consistent. Figure 4.7 thus shows the flow diagram for the example specification from Section 4.3.2. As previously mentioned, the complete specification consists of four components. Each component corresponds to one of the initial DRs (see Figure 4.5) and contains all design elements derived from it. Each component in an AD4ML flow diagram has a header and a *match* section with the codes of the design elements that belong to it. Any nested components are shown in the second *decomposition* section. Expanding these nested components recursively displays the same

layout as in the parent component, i. e., with a header, *match* and *decomposition* sections. DR-i-1, DR-p-3 and DR-o-4 have not been decomposed so far. Therefore, they are shown as one plain component, only with a header and the *match* section containing one DR code. Input component 2 contains the input components 2/1 and 2/2. These components group the design elements derived from DR-i-2 through zigzagging processes (see Step 4 in Figure 4.6 earlier). Specifically, the parent input component 2 contains DR-i-2, AC-i-2 and TR-i-2. The child input component 2/1 contains DR-i-2/1 and AC-i-2/1, and child input component 2/2 has DR-i-2/2 and AC-i-2/2.

$$\begin{pmatrix} DRi_1 \\ DRi_{2/1} \\ DRi_{2/2} \\ DRp_3 \\ DRo_4 \end{pmatrix} = \begin{bmatrix} A_{1,1} & 0 & 0 & 0 & 0 \\ 0 & A_{2/1,2/1} & 0 & 0 & 0 \\ 0 & 0 & A_{2/2,2/2} & 0 & 0 \\ 0 & 0 & 0 & A_{3,3} & 0 \\ 0 & 0 & 0 & 0 & A_{4,4} \end{bmatrix} \begin{pmatrix} ACi_1 \\ ACi_{2/1} \\ ACi_{2/2} \\ ACp_3 \\ ACo_4 \end{pmatrix} \quad (4.16)$$

There are two types of connections in a flow diagram (Suh, 2001). The summation junction connects uncoupled components. These are components that can be executed in any order as long as their results are ready at the junction point. The control junction connects decoupled components. These are components that need to be executed in a specific order to function. The types of design elements introduced in AD4ML make it easy to verify the logic of these junctions. They refer to the logic of data-driven machine learning. Input components 1 and 2 must prepare the training data before processing component 3 can learn on it. Also, processing component 3 must make a prediction before output component 4 can notify workers at the shopfloor. The summation junction between input components 2/1 and 2/2 joins the assessment of data significance and the data representation. This means that the two components can be executed in sequence or in parallel. The only condition is that preprocessing is finished before the learning process begins.

The AD4ML flow diagram can serve as an interactive digital document among stakeholders. It offers an intuitive overview of the team's progress in the design process. In [Figure 4.7](#), it is easy to identify which components are missing ACs and TRs. These are the ones displaying only a DR code in the *match* section. It is also easy to identify which components have not been decomposed yet. These components lack a *decomposition* section. After successive zigzagging processes, each component in the diagram will fill up. In that case, any component in the diagram can be expanded to *zoom in* into its contents. For example, clicking on input component 2/1 can reveal the design elements it contains: DR-i-2/1 and AC-i-2/1.

4.3.4. Reusability of Specification Components

The reusability of specification components is very relevant in the manufacturing domain. It improves the economic viability of designing and developing ML solutions. This subsection discusses how AD4ML enables the reuse of specifications components.

AD4ML specification components group all design elements describing an ML solution component. This includes all DRs, ACs and TRs derived from the same initial DR. If they have been decomposed, they also contain nested specification components.

For a specification component to be reusable, it must fulfill the *independence axiom*. This axiom requires specification components to be either uncoupled or decoupled. This can be verified using both design matrices. Compliant specification components can then be transformed into loose, reusable specification components.

The transformation of a reusable specification component adds sub-matrices of design matrices A and B corresponding to its design elements, as well as a reference to the original ML solution specification. Decomposed specification components need to be recursively transformed to extract its nested specification components. This recursive transformation of specification components makes it possible to recommend reusable specification components of different complexities.

For example, top input components, i. e., with an initial DR not referenced on any other reusable component, can be useful when designing a new ML solution to specify how to read data from a known source. Another example would be to use the most nested processing components, i. e., without any outward decomposition links, as reference to specify the training settings of a new ML model.

4.3.5. Agile Design of ML Solutions

Two characteristics of AD4ML support setting up the design process as an agile development project. First, components are specified across design layers, which relate to distinct roles. Second, each design layer contains design elements arranged in hierarchies. These hierarchies always expand via zigzagging one level at the time.

In an agile approach, design elements of the upper design layers become tasks for which design elements of the lower design layers have to be *developed* (i. e. *matched*). Team members specifying the elements of an upper layer thus adopt the agile role of product owners, whereas team members on a lower layer become the agile development team. The first sprint sets the scope of the ML solution specification by identifying all components with initial and abstract design elements in all design layers. This means the concurrent matching of DRs, ACs and TRs, triggering many zigzagging processes simultaneously. Subsequent sprints may decompose all or some of these design elements, one abstraction level at a time. During any given sprint, stakeholders need to specify DRs, ACs, and TRs for the same set of components. During the first sprints, the set may include all components. After several decomposition sprints, choosing which component set to decompose has to be decided at the sprint planning. The ML solution specification that emerges throughout sprint iterations and several decompositions can be seen as the product backlog in SCRUM (Schwaber and Sutherland, 2017).

This way of dividing the design process allows domain experts to contribute to the ML solution specification in a familiar manner. From their perspective,

they need to list down requirements as they would do for a specification sheet to commission the development of the ML solution.

4.4. Approaches to Validate and to Assess ML solution Specifications During the Design Process

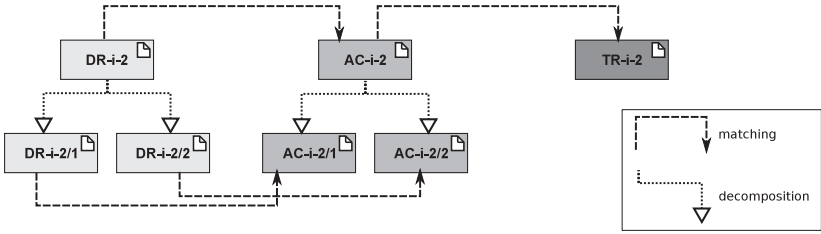
This section discusses two contributions that build upon the use of AD4ML in the design process. The section first discusses how AD4ML enables the validation of specifications while they are being designed. It then discusses how AD4ML provides ways to assess available specifications.

4.4.1. Validation of ML Solution Specifications

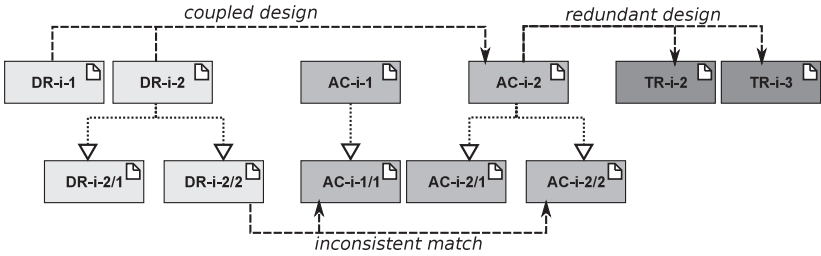
Typically, many ML solutions are designed, developed and tested using training and test data. The validity of their specifications is indirectly tested based on performance results. Any design flaws are noticed late, after the design process, when the ML solution is tested.

Instead, AD4ML embeds an axiomatic validation approach in the design process. This contribution allows a much earlier recognition of design flaws. Specifically, validation occurs during matching or decomposition steps. AD4ML validates design choices against the *independence* and *information axioms*. A comparison between the example specification of [Section 4.3.2](#) (see [Figure 4.8a](#)) and a flawed one (see [Figure 4.8b](#)) illustrates this contribution. The comparison shows when the axioms prevent bad design choices.

Validation first occurs during the matching step of the zigzagging process. In the example specification, the data scientist proposes a data preprocessing pipeline as AC-i-2 to match DR-i-2 (see [Figure 4.8a](#)). It is also assumed that the other initial DRs are matched to their own specific ACs. This situation leads to a one-to-one match between DR-i-2 and AC-i-2, as shown in [Equation \(4.9\)](#). All other pairs of initial DRs and ACs have one-to-one matches. This fulfills the *independence axiom*. However, the data scientist may want to use the same pipeline to fulfill DR-i-1. This DR is also of type input and states the need to collect all available data. Thus, the data scientist may



(a) Design elements derived from DR-i-2 after the four steps shown in Figure 4.6



(b) Design elements for DR-i-2 following a flawed design process

Figure 4.8.: Comparison between correctly and incorrectly designed ML solution specifications

want to match both DR-i-1 and DR-i-2 to AC-i-2, as shown in Figure 4.8b. Doing so produces the following design matrix A:

$$\begin{Bmatrix} DRi_1 \\ DRi_2 \\ DRp_3 \\ DRo_4 \end{Bmatrix} = \begin{bmatrix} A_{1,2} & 0 & 0 \\ A_{2,2} & 0 & 0 \\ 0 & A_{3,3} & 0 \\ 0 & 0 & A_{4,4} \end{bmatrix} \begin{Bmatrix} ACi_2 \\ ACp_3 \\ ACo_4 \end{Bmatrix} \quad (4.17)$$

This results in a coupled design, which violates the *independence axiom*. This specification works, but it is less robust and more complex than the example specification. Thus, the functionality for DR-i-1 may interfere with the functionality for DR-i-2, producing a performance bottleneck. For example, the collection of new data may need to wait until the preprocessing

of other previous data finishes. A design matrix with this form should be avoided by stakeholders, as no diagonal can be formed.

A similar situation occurs at the matching between AC-i-2 and TR-i-2. In the example specification, this is a one-to-one match (see Figure 4.8a). Scikit learn provides all functionality required by the data preprocessing pipeline. However, the software developer may prefer a combination of Python packages to materialize AC-i-2. For example, s/he can choose to replace Scikit learn with the SciPy and Pandas packages. These are represented by TR-i-2 and TR-i-3 in Figure 4.8b. This decision produces the following design matrix B:

$$\begin{pmatrix} ACi_1 \\ ACi_2 \\ ACp_3 \\ ACo_4 \end{pmatrix} = \begin{bmatrix} B_{4,4} & 0 & 0 & 0 & 0 \\ 0 & B_{2,2} & B_{2,3} & 0 & 0 \\ 0 & 0 & 0 & B_{3,4} & 0 \\ 0 & 0 & 0 & 0 & B_{4,5} \end{bmatrix} \begin{pmatrix} TRi_1 \\ TRi_2 \\ TRi_3 \\ TRp_4 \\ TRo_5 \end{pmatrix} \quad (4.18)$$

This produces a redundant design and hence conflicts with the *information axiom*. The implementation in Figure 4.8b works, but requires maintaining more software dependencies. This can mean an extra need to specify the interactions between the two packages. Similarly, future versions of these packages may become incompatible with each other. A design matrix in this form shows stakeholders where to improve the specification in order to create a diagonal.

The design choices to decompose design elements may also be validated easily. This prevents inconsistencies between decomposition levels. For example, DR-i-2 may be matched only to AC-i-2, as shown in Figure 4.8a. This one-to-one match respects the *independence axiom*. Even so, matches between their decomposed elements may become inconsistent. For example, the data scientist may want to match DR-i-2/2 with AC-i-1/1 and AC-i-2/2 (see Figure 4.8b). DR-i-2/2 demands data to be in a format easy to analyze. The match to AC-i-2/2, i.e., the use of data frames in the data processing pipeline, is consistent with the match at the upper level. Yet, the data scientist

may also want to match DR-i-2/2 to AC-i-1/1. S/he may wish to collect data directly in a data frame, instead of having to encode it later. However, AC-i-1/1 is the decomposition of AC-i-1, which was not matched with DR-i-2 (see Figure 4.8b). This becomes evident when comparing design matrix A before (Equation (4.19)) and after (Equation (4.20)) the decomposition

$$\begin{pmatrix} DRi_1 \\ DRi_2 \\ DRp_3 \\ DRo_4 \end{pmatrix} = \begin{bmatrix} A_{1,1} & 0 & 0 & 0 \\ 0 & A_{2,2} & 0 & 0 \\ 0 & 0 & A_{3,3} & 0 \\ 0 & 0 & 0 & A_{4,4} \end{bmatrix} \begin{pmatrix} ACi_1 \\ ACi_2 \\ ACp_3 \\ ACo_4 \end{pmatrix} \quad (4.19)$$

$$\begin{pmatrix} DRi_1 \\ DRi_{2/1} \\ DRi_{2/2} \\ DRp_3 \\ DRo_4 \end{pmatrix} = \begin{bmatrix} A_{1,1/1} & 0 & 0 & 0 & 0 \\ 0 & A_{2,1,2/1} & 0 & 0 & 0 \\ A_{2,2,1/1} & 0 & A_{2,2,2/2} & 0 & 0 \\ 0 & 0 & 0 & A_{3,3} & 0 \\ 0 & 0 & 0 & 0 & A_{4,4} \end{bmatrix} \begin{pmatrix} ACi_{1/1} \\ ACi_{2/1} \\ ACi_{2/2} \\ ACp_3 \\ ACo_4 \end{pmatrix} \quad (4.20)$$

As a result, the design equation for DR-i-2/2 includes a term with AC-i-1/1. This is inconsistent with the initial design equation between DR-i-2 and AC-i-2, which lacks a match to AC-i-1. The specification hence needs correction. Two alternatives are possible. The first one, the initial equation matches DR-i-2 to AC-i-1 too. The second one, the decomposition equation removes the match between AC-i-1/1 and DR-i-2/2.

This axiomatic validation allows AD4ML to recognize design flaws while the design process is taking place. Without AD4ML, validation needs to be an extra step in the design process. Otherwise, design flaws are discovered when the ML solution is developed or deployed.

4.4.2. Assessment of ML Solution Specifications

AD4ML enforces a structure to organize the contents of an ML solution specification via typed components. Each component has design elements with match and decomposition relationships between these elements. This

organizing structure enables another contribution of AD4ML to the design process. Namely, the assessment of ML solution specifications. Two types of assessment are possible. First, an assessment of the contents of the specification. Second, an assessment of many specifications to choose the best one.

An analysis of the structure gives insights about the contents of the ML solution specification. The example specification consists of four components, out of which two are of type input. This shows that the data input and preprocessing are more elaborate than the processing or output in this ML solution. Similarly, it shows that the ML solution is not very complex. It has four components when the minimal AD4ML specification has three, one of each type. The analysis of its structure may also determine which additional components can complement the specification. This can happen when an existing specification must provide new capabilities. A new output component can be added if more target systems need access to the predictions. A new processing component could provide reinforcement learning to handle concept drifts over time. Thanks to the *independence axiom*, it can be safely assumed that a higher number of components implies more complex ML solutions. Decomposition levels also offer an assessment of contents. ML solution specifications with more decomposition levels are by definition more detailed. They are thus closer to a finished specification that may be implemented. Note that a certain high number of decomposition levels does not guarantee a finished specification. This occurs only when the TRs at the lowest decomposition level are fully configured, available software. Yet, experienced practitioners can determine the number of decomposition levels that their specifications need. Without AD4ML, assessing the contents and complexity of an ML solution becomes more complicated.

Assume that several teams have developed many complete specifications for a particular ML task. Then, the two axioms provide clear assessment criteria to choose one. The *independence axiom* prefers uncoupled or decoupled specifications. This means specifications with one-to-one or at least one-to-many relationships between their design elements. This is because they result in less complex ML solutions. The *information axiom* prefers

specifications with a minimal number of design elements. This is because they produce more robust and non-redundant ML solutions. These solutions are easier to maintain and are more tolerant to variance in their operation. Assessment of specifications hence becomes a comparison of relationship cardinalities and of the number of design elements.

Without AD4ML, both types of assessments are more complicated. Documentation must be inspected and interpreted to assess the specification's contents and complexity. The specification may not be organized in independent components or have a consistent level of detail. If ML solutions are documented in different ways, it is difficult to find information for comparison. Moreover, common criteria to compare them has to be agreed upon.

4.5. Prototypical Implementation

A prototype by the name of ML solution designer showcases the usage of AD4ML to design specifications. The prototype was developed as a series of modules in python 3.8.3¹ and R 3.6.3². The modules form a python-based *web client* and an R-based *back-end API*. The module architecture of the ML solution designer is shown in [Figure 4.9](#). The back-end is a REST API that implements the operations to create and manipulate AD4ML specifications. The specification repository was implemented as two document collections in MongoDB. The first collection stores complete AD4ML specifications, whereas the second one stores reusable specification components. All parts of the prototype exchange specifications in JSON format.

Complete AD4ML specifications are formed by four objects. These are "Domain_Requests", "Analytical_Concepts", "Technological_Resources" and "Design_Matrix". The first three objects contain the design elements indicated by their name. Each design element is a separate object with the fields "code", "type", "title", and "description". Decomposed design elements are stored in a nested object inside their parent design element. This preserves

¹<https://www.python.org/downloads/release/python-383/>

²<https://cran.r-project.org/bin/windows/base/old/3.6.3/>

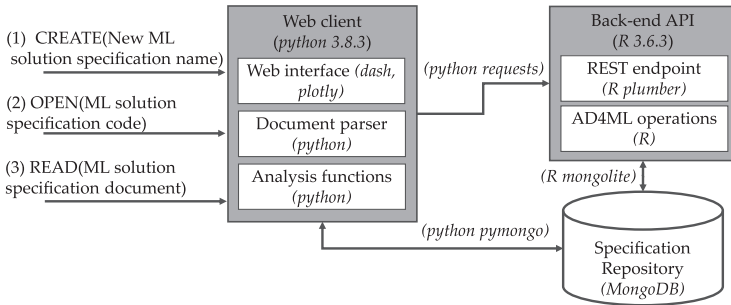


Figure 4.9.: Module architecture of the ML solution designer. The programming languages and libraries used are shown in parentheses.

the decomposition hierarchies built during the zigzagging processes. The object "Design_Matrix" contains all matches between any two design elements. It thus combines design matrices A and B. Matches are stored as JSON arrays.

Users can interact with the prototype in three ways (see Figure 4.9). First, they can create a new AD4ML specification by providing the use case name. This is the case when the design process is about to begin. Second, they can load an existing AD4ML specification by choosing it from the list of specifications stored in the repository. This may be the case, e. g., when the specification is finished and needs to be retrieved to be implemented. Third, they can drag and drop an AD4ML specification stored in JSON format into the web client interface for it to be rendered. This can be useful if the team wishes to share the specification with others without making it generally available in the repository. Figure 4.10a shows the three options available on the web client.

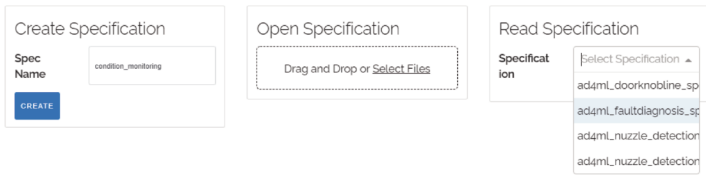
After picking one of the three cases, the prototype renders and analyzes the corresponding JSON specification. Assume that the specification for the fault diagnosis use case from Section 4.1.1 has been stored in the repository. Figure 4.10b shows the web client with the specification rendered on the interface. Aside from visualizing specifications, users can create, edit or delete individual design elements, as well as match and decompose one or more

design elements. The main section arranges the specification components in three columns corresponding to the *input*, *processing* and *output* component types. Any decomposed components can be displayed by expanding the parent component. This visualization implements parts of the AD4ML flow diagram (see [Section 4.3.3](#)). It depicts each specification component with a header, a *match* section and a *decomposition* section. The second section displays design elements that have been created but not linked to other design elements, either by a match or decomposition operation. This section can be thought of as a working area where design elements are collected and edited before being matched to a specific specification component.

After rendering the AD4ML specification, the ML solution designer analyzes the specification contents and informs users about the state of the specification. [Figure 4.11](#) shows the summary section in the prototype for the specification of the fault detection use case. The summary section lists the number of components that make the specification (see second row in [Figure 4.11](#)). It also states the number of times those components have been decomposed (see third row in [Figure 4.11](#)). This gives users a rough indication of how advanced is the design of the specification. The prototype also indicates users which components need additional design elements to be completed (see fourth row in [Figure 4.11](#)). The example in [Figure 4.11](#) shows that most components lack TRs, probably a sign that software developers have yet to contribute to the specification. If any of the axioms are violated, the corresponding text is shown in the design warnings row.

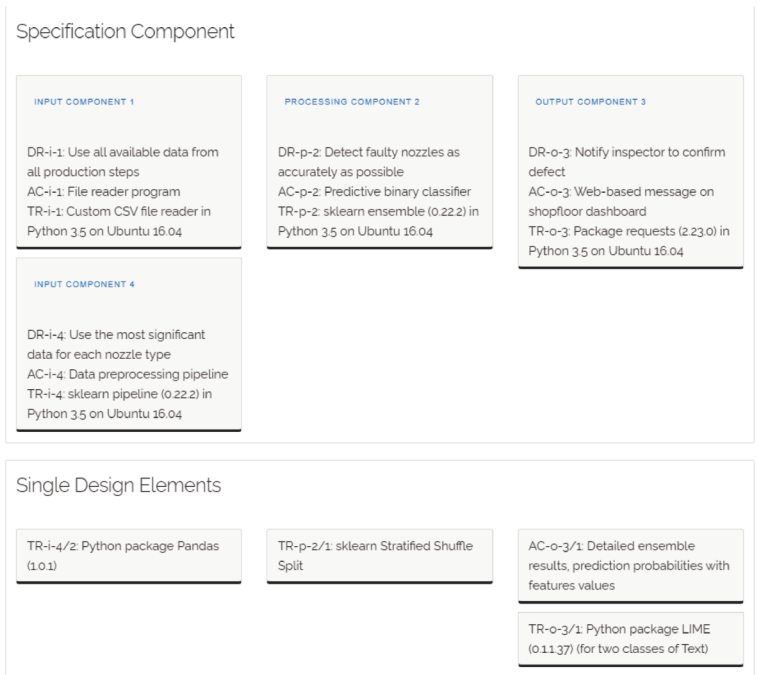
Finally, users can store specification components directly from the web client. This parses all specification components from the JSON document and reformats them as individual JSON documents as proposed in [Section 4.3.4](#).

ML Solution Designer



(a) Available interaction modes

ML Solution Designer



(b) AD4ML specification for the Fault Detection Use Case

Figure 4.10.: ML solution designer

Summary	
Info	ad4ml_faultdiagnosis_spec_01
Details	It contains 4 top level elements with 2 Inputs, 1 Processing and 1 Output
Specification	Specification has been decomposed 1 times
Component Missing Design Elements	Processing component 2/1: DR-p-2/1 is missing AC, TR Output component 3/1: DR-o-3/1 is missing AC, TR Input component 4/2: AC-i-4/2 is missing TR
Design Warnings	None

Figure 4.11.: Detail of the analysis text available on the ML solution designer

4.6. Discussion and Assessment

This section discusses how AD4ML addresses the feasibility requirements introduced in [Section 4.1.2](#). It describes the characteristics of AD4ML that address each requirement and determines to which extent it is fulfilled.

4.6.1. R_1 . Enablement of Systematic Experimental Learning

The matching between the *analytics* and the *technology-specific* design layers handles R_1 . The *analytics* design layer specifies a technology-independent conceptual architecture of all components of an ML solution. The *technology-specific* design layer produces a technical software specification for this architecture and its components. This clear division between conceptual architecture and technical implementation allows to easily design several completely or partially different implementations for the same conceptual architecture. For instance, TR-i-2 in [Figure 4.6](#) describes a Python implementation for AC-i-2. Other implementations in Java or R can be designed using the same AC.

Also, changes in functionality can be easily correlated with the ML solution's components or subcomponents. In the specification shown in [Fig-](#)

ure 4.6, there may be a need to change the kind of data used by the ML solution. Instead of using the most significant data (DR-i-2), the most frequent data may be required (DR-i-2'). Thanks to the match coefficients contained in the Design Matrices A and B, the impact and effect of this change is easy to identify. Design equation 4.9 shows that only AC-i-2 needs to be revised. Following the match relationships, design equation 4.12 shows that TR-i-2 is the only software element materializing AC-i-2. This indicates that only the scikit learn pipeline needs modifications to use the most frequent data. This analysis of the design equations also applies at deeper levels of decomposition.

4.6.2. R_2 . Clear and Objective Documentation of the Design Intention

AD4ML keeps a separate statement of the design intention with the DRs of the *domain-specific* design layer. Nevertheless, these DRs are decomposed along with the other design elements during the zigzagging process. This guarantees that the design intention is as detailed and current as the specification of implementation details of the ML solution in the ACs and TRs. As a result, even the purpose of specific settings, e. g., the use of a sampling strategy to obtain train and test data splits, can be known. Thus, it can be said that DRs provide the clear and objective documentation of the intention behind each software component, i. e., they fulfill R_2 .

In the example specification (see Figure 4.6), a domain expert with no knowledge of Machine Learning can still know that the scikit learn pipeline is the software component responsible for finding the most significant data. S/he can thus understand the effect that modifying or removing that scikit learn pipeline can have in the use case.

This level of detail opens the possibility to search for specifications made for the same or a similar *purpose*. This implies retrieving ML solution specifications with the same DRs, or with a common subset of them. Instead of making relative comparisons between them, the most suitable one can be selected based on the *information axiom*.

4.6.3. R₃. Support of End-to-end Traceability and Consistency

R₃ is addressed by AD4ML by means of the zigzagging process. The line of thought can be traced in the sequence of matching and decomposition steps across design layers. This trace is end-to-end because it goes from the initial design intention in the context of the use case to the most specific settings of the ML solution software components. It is all encoded in the design matrices, from the initial DRs to the most concrete TRs.

Having this traceability information facilitates the creation of a revised version of the ML solution. In the example specification, it is easy to find out why scikit learn is used. The match relationships between DR-i-2, AC-i-2 and TR-i-2 trace the decisions of domain expert, data scientist and software developer that lead to that choice (see [Figure 4.8a](#)). Each design element in that line of thought can be reviewed in case the functionality provided is insufficient. It can be determined, e. g., that AC-i-2, put forward by the data scientist, is inadequate. This means that DR-i-2 cannot be satisfied by TR-i-2, even if the latter is a good match to AC-i-2. Also, consistency is verified between decomposition levels. This allows to see if and where a compromise is made. The flawed specification in [Figure 4.8b](#) illustrates this. Equations [4.19](#) and [4.20](#) show an inconsistency in the way DR-i-2 is satisfied. The data scientist initially claims that the data preprocessing pipeline alone (AC-i-2) can satisfy DR-i-2. Later, s/he decides that AC-i-1, through its decomposition DR-i-1/1, is also needed. In this state, the decomposition is inconsistent and hence a compromise at either level must be done.

4.7. Related Work

This section compares AD4ML with related approaches concerning their ability to address the requirements in [Section 4.1.2](#). A fundamental approach in this regard is the CRISP-DM model (Chapman et al., 2000). Additionally, the related work also surveys Continuous QFD (Herzwurm, Schockert, et al., 2015) and a conceptual modeling framework to design analytical

solutions (Nalchigar and Yu, 2020). Table 4.2 summarizes the related work observations.

CRISP-DM. The Cross-industry Standard Process for Data Mining is the de-facto standard for practitioners to carry out data mining projects in domain-specific use cases. It provides an industry-, application-, and tool-neutral reference process model with guidelines to analyze data and gain knowledge from it (Shearer, 2000). The six phases of the process model provide a sequence in which tasks should be carried out, as well as certain loops from one phase back to previous phases whenever modifications are needed (Chapman et al., 2000).

Regarding the design of ML solutions, CRISP-DM focuses on setting up a project plan. The contents of the project plan overlap with those of the ML solution specification in AD4ML. For instance, business objectives relate to the design intention. However, the project plan is rather meant to guide iterations of the CRISP-DM cycle. It is updated in every cycle iteration until an *empirically approved* specification emerges. In the end, the ML solution specification is the sum of all documentation created throughout the phases.

Overall, CRISP-DM partially enables experimental learning. Its iteration approach and the documentation allow for testing different software components. However, several factors complicate experimental learning. Cycle iterations must go through the same phases regardless of the changes pursued. It is not easy to determine the impact of changing one component. This is because dependency information is scattered in the project plan and among all other documents created during the phases.

The business objectives and success criteria are relevant for the clear and objective documentation of the design intention. They describe the domain-specific purpose of the ML solution. Nevertheless, this information is mixed with project information in the project plan, while AD4ML allows for a clear and concise presentation of only the ML solution's objectives.

The end-to-end traceability and consistency of the ML solution specification is not addressed by CRISP-DM. It does not keep clear track of the

Table 4.2.: Coverage of feasibility requirements by related approaches. ++ shows complete fulfillment, + shows partial fulfillment, - shows non-fulfillment.

Feasibility Requirement	CRISP-DM	Continuous QFD	Conceptual modeling framework ...
R₁ . Experimental learning	+	-	-
R₂ . Clear design intention	+	++	++
R₃ . End-to-end traceability	-	-	-

dependencies between the software components and the business objectives they are supposed to fulfill. For this reason, a general and sophisticated review of the process is needed during the evaluation phase. Checking the consistency at this stage requires manual inspections of all the project documentation.

Continuous QFD. Quality Function Deployment (QFD) is a requirements engineering method that captures customer needs and transforms them into product requirements. Different adaptations of QFD have been proposed for specific application domains. Continuous QFD (CQFD) is an adaptation for highly-dynamic software development (Herzwurm, Schockert, et al., 2015). It is meant to handle unclear customer requirements and changing software components (Herzwurm, Dowie, et al., 2001). An integral part of CQFD are content templates. These templates represent the adaptation of QFD concepts to the context of an innovative technology, e. g., cloud computing. They contain a predefined set of customer requirements and software characteristics (Herzwurm, Dowie, et al., 2001). These two types of elements are set in a matrix to document their dependencies. So, the template captures the equivalent of DRs and ACs, as well as the design matrix capturing dependencies between DRs and ACs. This implies that CQFD provides a clear documentation of a design intention.

Nevertheless, CQFD fails to enable experimental learning specifically because of the use of content templates. The predefined set of software characteristics offered by the templates limits the ability to test software components. Any characteristic not present in the template cannot be included in the ML solution, even if it fulfills customer requirements. Additionally, CQFD cannot offer end-to-end traceability and consistency. This is because it stops when software requirements are specified, which are similar to the conceptual architecture defined by the ACs. So, CQFD does not offer any means to specify the technical software implementations. In contrast, AD4ML provides these means via TRs and the design matrix B to capture dependencies between ACs and TRs.

Conceptual Modeling Framework for Business Analytics Solutions.

This framework is composed of three views: *business*, *analytics design* and *data preparation* (Nalchigar and Yu, 2020). These views roughly correspond to the *domain-specific* and *analytics* design layers of AD4ML. The difference lies in the way this framework captures the dependencies between its layers. All three views are connected solely through *analytics goals*. These are elements in the *analytics design* View. They capture the type of analysis to be performed over the data set, i. e., prediction, prescription or description (Nalchigar and Yu, 2020).

This framework addresses the second requirement in that it can document the design intention using its *business view*. However, having the *analytics goals* as the sole connector between views makes it difficult to capture all the component dependencies. For example, there is no way to indicate if the choice of an algorithm needs a particular data transformation. This hinders experimental learning, because it requires that practitioners manually determine the impact of changing components. It also complicates checking the consistency from a design intention to the implemented ML solutions. So, the framework cannot support end-to-end traceability and consistency.

4.8. Summary and Future Work

This chapter introduced AD4ML, an adaptation of Axiomatic Design for the specification of ML solutions in manufacturing. The chapter discussed the rationale behind the adaptation and illustrate this by designing an example specification for a fault detection use case. It also explained three improvements that AD4ML brings to the design process. They include the visualization of specifications in AD4ML flow diagrams; the reutilization of specification components and the agile design of specifications. Together, they ease the design effort of the stakeholders. The chapter also discussed how the use of AD4ML can be leveraged to enable the validation and assessment of ML solution specifications. This allows for the early identification of flawed design choices. Stakeholders can thus correct them before developing or deploying the whole ML solution. The chapter presented a prototypical implementation of AD4ML in the form of the ML solution designer. The prototype depicts the advantages of designing ML solutions with AD4ML. Namely, it enables the visualization and design of specifications, it provides analysis information on the specification contents, and it allows the consolidation of reusable specification components in a repository. Furthermore, the chapter discussed how AD4ML addresses the feasibility requirements faced by the process of specifying ML solutions. It does this by providing clear documentation of the design intention, the conceptual and technical solution specifications, as well as linkages between them.

A future research direction may be to automatically generate design recommendations based on existing ML solution specifications. For that, the observed performance of implemented ML solutions must be documented and bound to components in their ML solution specification. These bindings may help find correlations between good or bad ML solution performance and the components that yield that performance. Bundled with a repository of reusable components (see [Section 4.3.4](#)), a recommender system can be developed. This can make it easier for less experienced users to participate in the design process. The recommender system in the next chapter represents

a first step in this direction, which nonetheless lacks the integration with AD4ML specifications.

CHAPTER 5

ASSISTML

The sheer number of possible ML solutions increases the complexity to select suitable solutions for a new use case (see *selection challenge CH-3* in [Section 1.3](#)). Addressing this complexity is difficult for citizen data scientists or domain experts, who are becoming more and more involved in data science projects (Gröger, 2018). These practitioners have limited knowledge of ML solutions, but in-depth understanding of the use case. Furthermore, citizen data scientists face practical requirements when building ML systems, which go beyond the known challenges of ML, e. g., data engineering or parameter optimization, and span over the complete development process (see *process challenge CH-4* in [Section 1.3](#)). For instance, they are expected to quickly identify ML system options that strike a suitable trade-off across multiple performance criteria. These options also need to be understandable for non-technical users like them, i. e., recommended ML solutions should explain their suitability. Citizen data scientists are also encouraged to reuse components from existing ML solutions to reduce development time and costs. Addressing these practical requirements represents a problem for citizen data scientists with limited ML experience. This calls for a method to help them identify suitable ML software combinations.

Different approaches, such as AutoML (Vanschoren, 2018), Explainable AI (Burkart and Huber, 2020), and Meta-learning (Vanschoren, 2018), have been proposed to tackle the problem of selecting and configuring software components for ML solutions (see [Section 5.2](#)). Yet, they address these requirements only partially.

This chapter introduces AssistML, a concept to recommend ML solutions for predictive use cases. This concept targets citizen data scientists, enabling them to apply ML solutions without the involvement of more experienced data scientists. The concept uses performance preferences provided by citizen data scientists and matches them to existing metadata about ML solutions to facilitate the selection and configuration of ML components. Furthermore, it offers intuitive explanations of the recommended ML solutions via a recommendation report.

The approach was implemented and evaluated based on two *new* data sets, i. e., not contained in the metadata repository. This evaluation shows that AssistML leads to less-complex ML solutions that meet the user's preferences considerably faster than the AutoML system. Moreover, the recommended ML solutions clearly show their trade-offs across multiple performance criteria and provide concise and relevant information to the users.

The contents of this chapter are an edited version of a separate publication (Villanueva Zacarias, Weber, et al., 2021). All concepts extracted from it and used in this dissertation are the original work of the dissertation author.

The remainder of this chapter is structured as follows: [Section 5.1](#) presents an application scenario and the results of a literature review to derive requirements for a system recommending ML solutions. With respect to these requirements, the chapter discusses related work in [Section 5.2](#). Then, [Sections 5.3](#) and [5.4](#) present *AssistML*. Finally, [Section 5.5](#) discusses the evaluation results, whereas [Section 5.6](#) concludes the chapter.

5.1. Application scenario for ML Solution recommendations

The first part in this section describes the exemplary application scenario in which practitioners benefit from recommended ML solutions. The second part discusses the practical requirements for a concept to provide such recommendations.

5.1.1. Reusing ML Solutions for Predictive Use Cases

Figure 5.1 depicts an exemplary scenario where an ML solution is to be developed for a new predictive use case. A team of practitioners is assigned to this task, e. g., a decision maker, a domain expert, a software developer and a citizen data scientist. In this context, citizen data scientists are particularly attractive when experienced ML experts are scarce (Flaounas, 2017). The team has access to the data of the new predictive use case and to a repository populated with metadata about existing ML solutions, some of which have been developed for similar use cases. Such repositories are part of various ML tools (Zaharia et al., 2018). So, they are available in many ML projects and organizations.

However, the reuse of ML solutions from the repository proves to be difficult. The citizen data scientist knows that there exist different languages, libraries, data formats, and deployment models to develop the new ML solution. Yet, besides his or her general ML knowledge, s/he does not know which of the existing ML solutions were developed for a similar use case. Thus, s/he has to determine for each individual ML solution in the repository whether the combination of software components can be reused in the new use case. For example, s/he has to verify that the software components can process the new use case data. S/He also has to validate that the performance of each existing ML solution meets the requirements of the new use case. This implies the consideration of performance trade-offs, e. g., preferring ML solutions with a faster prediction speed over others with higher prediction accuracy.

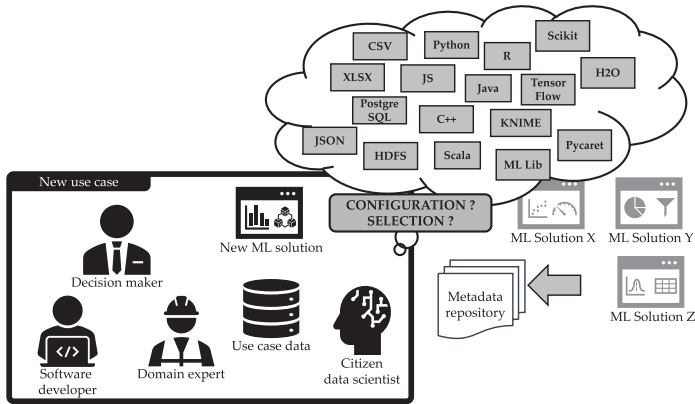


Figure 5.1.: Application scenario for recommendations of ML solutions.

5.1.2. Practical Requirements

The application scenario illustrates the complexity of reusing ML solutions and the human effort involved. To address this problem, citizen data scientists need concepts that recommend suitable ML solutions. This subsection discusses four key practical requirements that a concept for recommending ML solutions must fulfill. These requirements are based on a literature review and on discussions with industry partners.

[R₁] Reusability. Existing ML solutions should be the basis to make recommendations (Paleyes et al., 2020). As the number of developed ML solutions in an organization increases, it is more likely that existing ML solutions have configurations similar to the solution needed in the new use case. Citizen data scientists may thus want to reuse those existing ML solutions. This can mean either to reuse the implemented ML solution *as-is* or to consider the combination of ML components and their configurations as *blue-print recommendation* for a new implementation. To address this consideration, a common set of metadata about each ML solution has to be stored in a central metadata repository. These metadata need to ensure the

complete reproducibility of the ML solution so that the recommendation is applicable in new use cases.

[R₂] Explainability. Recommended ML solutions should be interpretable (Baier et al., 2019; Burkart and Huber, 2020; Ethayarajh and Jurafsky, 2020). The lack of experienced ML experts increases the need for explanations that are especially tailored to non-technical people such as citizen data scientists (Baier et al., 2019). Citizen data scientists prefer understandable ML solutions with acceptable performance over complex black-box ML solutions with state-of-the-art performance (Ethayarajh and Jurafsky, 2020; Paleyes et al., 2020). To address this consideration, meaningful explanations have to be generated for each recommended ML solution. Meaningful explanations focus on the effects of ML solution components on prediction results instead of focusing on standard ML metrics (Baier et al., 2019). They facilitate citizen data scientists to select a recommendation out of many that match his or her performance preferences (Ethayarajh and Jurafsky, 2020).

[R₃] Responsiveness. The process to generate recommendations should be responsive (Baier et al., 2019; Bernardi et al., 2019; Flaounas, 2017; Zaharia et al., 2018). ML solution development is an iterative process. Thereby, citizen data scientists test different hypotheses to assess whether an ML solution fulfills the use case needs (Bernardi et al., 2019; Flaounas, 2017). A recommendation system hence should suggest several ML software combinations to citizen data scientists, so that they may compare the solutions and select the most suitable one (Zaharia et al., 2018). Throughout this iterative search, citizen data scientists' preferences develop too, as they narrow down suitable ML solutions. However, iterations and the number of ML solutions citizen data scientists may assess are constrained by the resources allocated to the use case (Flaounas, 2017). In order to reduce time and costs for the search of ML solutions, it is important to provide recommendations in a responsive manner.

[R₄] Multi-criteria trade-offs. The suitability of ML solutions has to be decided based on multiple criteria (Baier et al., 2019; Bernardi et al., 2019; Ethayarajh and Jurafsky, 2020; Wagstaff, 2012). In practical use cases, the suitability of an ML solution depends on multiple factors instead of a

single ML quality metric, e. g., only accuracy or [Root Mean Square Error \(RMSE\)](#) (Wagstaff, 2012). For example, a very accurate ML solution may not be useful if it takes too long to make predictions or if it requires a lot of data and complex data preprocessing (Breck et al., 2017). In that case, a less accurate, but faster ML solution that requires less data can be a better recommendation (Baier et al., 2019; Bernardi et al., 2019; Ethayarajh and Jurafsky, 2020). Similarly, excessive computing infrastructure investments or many software dependencies may render a complex ML solution unsuitable for the new use case. It is important to generate recommendations with the consideration of multiple criteria and objectives. Furthermore, it has to be possible to identify and compare the trade-offs between the individual recommended solutions. For this purpose, information highlighting these trade-offs have to be made available to citizen data scientists.

5.2. Related Work

This section discusses three related approaches for ML solution development: AutoML systems, Meta-Learning, and Explainable AI. The related work assesses whether each approach fulfills the practical requirements from [Section 5.1.2](#). A summary of the analysis is shown on [Table 5.1](#).

5.2.1. AutoML Systems

Automated machine learning (AutoML) systems generate an optimized supervised learning model or pipeline (Feurer, Eggenberger, et al., 2020; Xin et al., 2021). The final ML model is optimized to, e. g., minimize generalization error or maximize accuracy. Open source examples of AutoML systems include Auto-sklearn (Feurer, Eggenberger, et al., 2020) or TPOT (Olson, Urbanowicz, et al., 2016).

For each new use case, AutoML systems search for high-performing ML models in a predetermined configuration space, consisting of learning algorithms, (hyper-)parameters and basic feature engineering (Xin et al., 2021). So, AutoML systems are not based on ML solutions that already exist in

an organization or on metadata describing these solutions in a repository. Therefore, they do not fulfill requirement R_1 .

AutoML systems also do not fulfill R_2 . They tend to produce complex ML models that citizen data scientists usually fail to interpret. They even prefer complex to simpler models if the complex models are only marginally better in the considered optimization objective (Xin et al., 2021). Furthermore, AutoML systems do not offer adequate explanations to citizen data scientists.

AutoML systems are not responsive at all (R_3), as they are known to be resource- and time-intensive (Xin et al., 2021). Even if the system is constrained by a time budget, the runtime to obtain a single optimized ML model can take up to 60 minutes (Feurer, Eggensperger, et al., 2020). Citizen data scientists are usually not willing to wait this long time, i. e., they want to assess different configurations of ML solutions in a much faster pace.

Fulfilling R_4 requires the consideration of multiple criteria. AutoML systems do not support this by design, since optimization strategies are executed to improve the value of one single evaluation metric, e. g., cross-validation loss (Feurer, Eggensperger, et al., 2020) or area under the operator curve (Gijssbers et al., 2019). So, it is not possible to consider any trade-offs between several optimization criteria and objectives with AutoML systems.

5.2.2. Meta-Learning

Meta-learning refers to data-driven approaches to learn from previously implemented ML models or pipelines (Vanschoren, 2018). Meta-learning approaches can be used to recommend ML models or their settings for a new task and data set (Pan and Yang, 2010). Metadata about the configuration, used training data, and observed performance of already existing ML models form the basis for meta-learning approaches (Vanschoren, 2018). Several platforms have been proposed to collect these metadata (Vanschoren et al., 2014; Zaharia et al., 2018). These platforms collect metafeatures about the data sets, the learning task, e. g., classification, the used (hyper-) parameters and data preprocessing techniques. In addition, they store results of evaluation metrics. Meta-learning platforms thus fulfill R_1 .

Table 5.1.: Assessment of related approaches

	AutoML	Meta-Learning	Explainable AI
[R ₁] Reusability	○	●	○
[R ₂] Explainability	○	○	◐
[R ₃] Responsiveness	○	○	◑
[R ₄] Multi-criteria trade-offs	○	◐	○

For meta-learning approaches to fulfill R_2 , they must be able to explain why an ML model suits the new data set. Explanations have to avoid the use of ML metrics that non-expert citizen data scientists cannot understand. Most meta-learning approaches base their selection on complex metrics, e. g., covariances of text feature pairs (Raina et al., 2006) or data complexity measures (Biondi and Prati, 2015). Using these metrics, each approach estimates the similarity between the new data set and the data sets used for already existing ML models. However, it is hard for citizen data scientists to understand these complex metrics and to reproduce how they determine the similarity of data sets. Thus, R_2 remains unfulfilled.

Regarding the responsiveness requirement R_3 , meta-learning approaches are expected to find various suitable ML models in a short time. Yet, only single meta-learning approaches meet this requirement, as they are limited to a specific type of algorithm (Biondi and Prati, 2015; Raina et al., 2006; Van Rijn and Hutter, 2018). For example, the approach of Raina et al. (Raina et al., 2006) is designed exclusively for text classification with logistic regression. Similarly, the approach of Biondi and Prati (Biondi and Prati, 2015) is designed exclusively for support vector machines. However, to efficiently use such meta-learning approaches independently of the underlying ML algorithm, a citizen data scientist has to invest additional time. To this end, s/he needs to execute and compare different meta-learning approaches in terms of their predictive quality and performance. This is error-prone and time-consuming. R_3 is thus not fulfilled.

Meta-learning approaches evaluate performance towards a single performance metric at a time, usually accuracy (Biondi and Prati, 2015) or classification error (Raina et al., 2006). Data similarity can only be indi-

rectly considered in the selection of suitable ML models (Biondi and Prati, 2015). Thus, meta-learning approaches may not consider more than one performance metric and one similarity metric at a time. They only partially fulfill the consideration of multiple criteria and trade-offs between them, as required by R_4 .

5.2.3. Explainable AI

Explainability approaches can describe the general functioning of a ML model (Burkart and Huber, 2020), i. e., give *global* explanations, or the reasoning behind an individual prediction, i. e., give *local* explanations. Since ML solution development is guided by the overall behavior of the solution, the discussion focuses on global explanations. Here, model-agnostic global explainers can be used with different supervised learning algorithms. These kinds of explainers only use the data set of the ML model they are applied to. They do not establish comparisons with other data sets or with other ML models. Thus, R_1 is not given.

Regarding the explainability requirement (R_2), model-agnostic, global explainability approaches provide two types of explanations. The first type are visualizations (Adler et al., 2018; Goldstein et al., 2015), e. g., partial dependency plots. These approaches plot the effect of one or multiple data features on the model's performance, e. g., on accuracy. Most plots are only able to display one or two data features at a time, rendering them impractical for ML models with many data features. The plots also display metrics such as obscurity (Adler et al., 2018) in unfamiliar formats, e. g., in various diverging and falling curves. These plots are difficult to interpret for non-experts. The second type of explanations are relevance metrics. These metrics quantify the impact that the presence or absence of a data feature has on the model. Example metrics are attribute interactions (Henelius et al., 2017) or attribute weights (Subianto and Siebes, 2007). Similar to the plots, the interpretation of these metrics is difficult for non-experts. In both cases, the interpretation of ML models depends on the use of explainability concepts, which have to be interpreted as well. Citizen data scientists however usually do not

completely understand these explainability concepts. Thus, explainability approaches partially fulfill R_2 .

To fulfill the responsiveness requirement, explainability approaches must provide explanations in short execution times. However, approaches such as *leave-one-out* (Burkart and Huber, 2020) or Explainer global (Subianto and Siebes, 2007) need to iterate multiple times over the data features. This is time-consuming when a big amount of features needs to be handled. Therefore, explainability approaches partially fulfill R_3 .

Finally, none of the approaches allows the consideration of multiple criteria when offering explanations. Explanations are offered for one performance aspect at a time. For instance, GFA plots obscurity versus accuracy (Adler et al., 2018). In ASTRID, attribute interactions are computed with respect to accuracy (Henelius et al., 2017). These approaches do not allow the consideration of trade-offs, e. g., between accuracy and training time. Thus, R_4 is not fulfilled.

5.3. AssistML Metadata Repository

AssistML provides its recommendations based on a centralized repository that contains metadata, source code, and sample training and test data of previously developed ML solutions. Metadata and source code are accessible via an identification code assigned to each ML solution, e. g., "*DTR-faultdetection-001*". The identification code describes in three parts (1) the learning algorithm with an abbreviation ("DTR"), (2) the use case ("faultdetection"), and (3) a sequence number for the specific combination of learning algorithm and use case. Data sets used to develop an ML solution are referenced in the solution metadata itself. The contents of the source code and data sets are specific for their use case. As such, their format and contents are determined before being submitted to the repository. This section defines the metadata schema that AssistML requires.

Metadata can be extracted directly from ML solution's software or from its documentation. This way, the repository helps to standardize different

metadata, so as to make all ML solutions comparable with each other. Let m_i be the metadata describing the i -th ML solution in the metadata repository M . Each entry m_i contains four metadata subsets:

$$m_i = \{u_i, d_i, s_i, p_i\} \quad (5.1)$$

$$u_i = \{taskType_i, taskOutput_i, deployment_i\} \quad (5.2)$$

$$d_i = \{allMF_i, singleMF_i, preprocessing_i, featsUsed_i\} \quad (5.3)$$

$$s_i = \{language_i, platform_i, algorithm_i, hParams_i\} \quad (5.4)$$

$$p_i = \{metrics_i, metricsLabels_i, margins_i\} \quad (5.5)$$

The **use case set** u_i firstly describes the *type of analytics task* the ML solution i performs. This refers to supervised learning tasks, e. g., binary or multi-class classification (Sokolova and Lapalme, 2009). The *task output* indicates the kind of result the ML solution produces, e. g., whether it delivers single predictions or class probabilities for a new observation. Finally, the *type of deployment* describes how the solution may be used in production environments. Examples can be deploying the ML solution in a cluster or as a stand-alone program on a single host. These metadata can be obtained from technical and project documentation about the ML solution development, e. g., in software specifications.

In $allMF_i$, the **data subset** d_i contains descriptive metrics, i. e., simple meta features (Vanschoren, 2018) about the complete data set for which an ML solution is developed. Different data sets can thus be compared on the basis of these metafeatures. They are computed on the original data set prior to the data preprocessing required by the ML solution. Examples include the total number of instances, the number of features, and percentages describing the shares of each feature type in the data set. Each data feature can belong to one of four feature types: numerical, categorical, datetime or unstructured text. $singleMF_i$ is a vector of length j that describes each data feature j with additional metafeatures specific to its feature type. Table 5.2 gives examples of the metafeatures that can be computed for each feature

type. Note that the list is not exhaustive and thus metafeatures can be added or removed for each feature type. Metadata in $preprocessing_i$ describe the used data preprocessing techniques. For each feature type in d_i , two lists indicate the techniques applied for the ML solution to be able to read the data (encoding) and to select features (selection). $featsUsed_i$ indicates which data features from the data set d_i are finally used by the ML solution.

Table 5.2.: Exemplary metafeatures by feature type

Feature	Description
Numerical	Number of outliers, number of missing values.
Categorical	Number of categories, imbalance ratio (count of the most frequent category over the count of the least frequent category)
Datetime	Minimum and maximum deltas between chronological timestamps, frequencies of months and days of the week.
Unstructured text	Relative vocabulary size, Shannon’s entropy (Bank et al., 2012). A <i>bag-of-words</i> representation (Sebastiani, 2002) with minimal text preprocessing is required (word-tokenization, lowercasing, stopword and punctuation removal)

The **technical settings** s_i describe the configuration and parameters needed to reproduce the ML solution. The metadata for this set can be collected directly from the ML solution code via custom annotations (Zaharia et al., 2018). Metadata in this set include lists with the programming language(s) ($language_i$), and the software platform(s) and libraries used ($platform_i$). Furthermore, $algorithm_i$ specifies the ML algorithm implementation used by the ML solution, e. g., `sklearn.naive_bayes.GaussianNB`. Custom hyperparameter values are stored in $hParams_i$.

The **performance set** p_i contains performance values and explanations of the ML solution. Three groups of metadata are collected. The first group, $metrics_i$, contains values of performance metrics obtained while testing the ML solution on a test set. For instance, $metrics_i$ can capture the accuracy,

precision, and recall values for a classifier. The values in $metrics_i$ are scaled to a range [0,1], going from worst or slowest performance to best or fastest performance.

The second group, $metricsLabels_i$, contains a performance label for each metric based on the values in $metrics_i$. The performance labels intuitively indicate non-experts how good a solution is in comparison to others. Performance labels are letter grades going from A+ to E. Grade A+ denotes the best values. Grades A to E denote five groups, each covering 20% of the value range of that performance metric among ML solutions. For fairness, these labels only compare ML solutions developed for the same task and data set. To determine the threshold values for each grade, 5 quantiles are computed.

The third group, $margins_i$, includes margin heuristics for all data features. A margin heuristic for a particular feature assesses how useful data feature variations are for an ML algorithm to perform its task. In case of a classification, the margin of a data feature estimates the differences between the values of correctly classified observations from those of incorrectly classified observations. In general, the margin heuristic for a data feature in binary classification tasks is:

$$margin_{feat} = \sum (w_1 \cdot margin^P, w_2 \cdot margin^N) \tag{5.6}$$

where $margin^P$ is the difference between true positives (TP) and false positives (FP), $margin^N$ is the difference between true negatives (TN) and false negatives (FN), and w_1 and w_2 the factors for a weighted average.

Depending on the feature type, the margin heuristic is computed differently. For numeric data and datetime deltas:

$$margin^{Pnum} = \frac{\sum_1^n |\overrightarrow{TPfeat} - \overrightarrow{FPfeat}|}{avg(avg(TPfeat), avg(FPfeat))} \tag{5.7}$$

where \overrightarrow{TPfeat} are the sorted data feature values in the true positives sample, \overrightarrow{FPfeat} are the sorted data feature values in the false positives sample and $avg(TPfeat)$ and $avg(FPfeat)$ are the average values of the same samples.

For categorical and unstructured text data:

$$margin^{pcat} = \sum \left| \frac{|levels(TPfeat)|}{|TPfeat|} - \frac{|levels(FPfeat)|}{|FPfeat|} \right| \quad (5.8)$$

where $|TPfeat|$ is the number of elements in the true positives sample and $|FPfeat|$ represents the number of elements in the false positives sample. $|levels(TPfeat)|$ is the count of each distinct data feature value in the true positives sample, e. g., how many times does the value *medium* appear in the data feature *size* in the TP sample; $|levels(FPfeat)|$ is the count of each distinct data feature value in the false positives sample. In general, the function *levels()* is equivalent to the output of the *COUNT()* function with the *DISTINCT* clause in SQL.

The margin heuristic is used by AssistML to explain the functioning of a recommended ML solution (see [Section 5.4.4](#)). A high value for instance may show that a data feature has very different values in true positive and false positive samples. This big difference indicates that the feature helps improve the classification precision.

5.4. AssistML: A Concept to Recommend ML Solutions

This section explains the four main steps that constitute AssistML. They recommend a short list of ML solutions for a new use case n based on meta-data from the repository M . AssistML requires three inputs (see [Figure 5.2](#)). These inputs form a query q_n to obtain a list of recommended ML solutions in $report_n$. Thereby, q_n requires only the absolute minimum information, making the query easier to determine for citizen data scientists. The first input in q_n is a description of the desired use case task u_n . The task may be described, e. g., as binary classification with a single prediction as output.

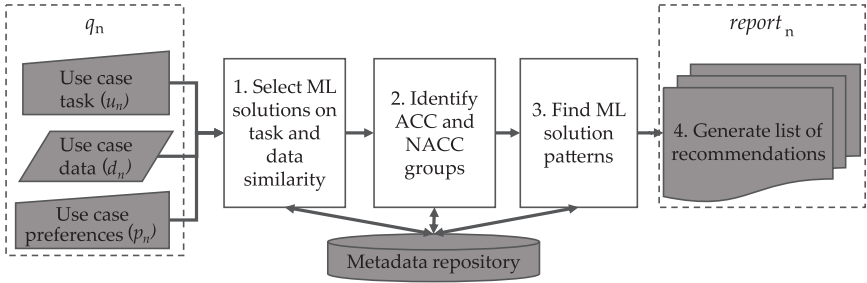


Figure 5.2.: Overview of the steps in AssistML

The second input is an annotated extract of the new use case data d_n . Citizen data scientists use annotations to indicate the feature types contained in the data, i. e., which of the four feature types numerical, categorical, unstructured text, or datetime are present. As third input, citizen data scientists set preferences they have about the ML solution’s performance (p_n). These preferences include ranges for performance metrics that define acceptable values for citizen data scientists. For instance, a classification task can have range limits for accuracy, precision, recall, and training time. A range of 0.25 for all metrics means that only ML solutions with values in the top 25% in all metrics are considered acceptable.

$$q_n = \{u_n, d_n, p_n\} \quad (5.9)$$

$$u_n = \{taskType_n, taskOutput_n\} \quad (5.10)$$

$$d_n = \{data_n, featTypes_n\} \quad (5.11)$$

$$p_n = \{ranges_n\} \quad (5.12)$$

5.4.1. Step 1: Select ML Solutions on Data Similarity

The first step selects ML solutions based on the similarity of the data set and task of existing ML solutions in M to the data set and task of the new use case in query q_n . Only the solutions developed for the most similar data sets

Algorithm 5.1 Select solutions on data similarity

Require: M, u_n, d_n **Ensure:** m_s, w_t

```
1:  $\{allMF_n, singleMF_n\} \leftarrow$   
    $analyzeQueryData(d_n.data, d_n.featsTypes)$   
2: function SELECT( $u_n, d_n$ )  
3:    $m_{s0} = \{m_i \in M \mid$   
      $m_i.taskType = u_n.taskType$  and  
      $m_i.taskOutput = u_n.taskOutput\}$   
4:   if  $m_{s0} = \{\}$  then  
5:     terminate ASSISTML  
6:   else  
7:      $\{m_s, w_t\} \leftarrow CHECKSIMILARITY(m_{s0},$   
      $d_n.allMF, d_n.singleMF)$   
8:     return  $\{m_s, w_t\}$   
9:   end if  
10: end function
```

and tasks, i. e., m_s , are passed to the following steps. The goal is to ensure that the recommended ML solutions were developed for a use case that is as similar as possible.

[Algorithm 5.1](#) gives an overview of the first step. It requires access to the repository M , and the subsets u_n and d_n of the user query. The first task is to analyze the provided data. The sample data $data_n$ and its feature type annotations are used to compute summary metafeatures $allMF_n$ and individual metafeatures $singleMF_n$ for the new use case data. These metafeatures allow the determination of similarity between the new data set d_n and existing data sets d_i in M .

Data set similarity is expressed in four levels. Each similarity level is defined by criteria regarding the use case task and data set. Firstly, the base similarity or m_{s0} (see [Algorithm 5.1](#)) describes ML solutions developed for the same type of task and same type of output. For example, m_{s0} can refer to all solutions in M for binary classification ($taskType$) and producing single predictions ($taskOutput$). If no solutions in M fulfill the criteria, the whole recommendation process is terminated ([Algorithm 5.1](#)). This is indicative

of a completely new use case, for which none of the solutions in M can be reused. In that situation, other development approaches should be followed.

If solutions with similarity level 0 are available in M , the function *check Similarity()* applies filter criteria to determine which solutions fulfill levels 1 to 3. The filter criteria for each level are given in Equation (5.13), Equation (5.14) and Equation (5.15). The similarity levels are consecutively checked, so that $m_{s3} \subseteq m_{s2} \subseteq m_{s1} \subseteq m_{s0}$.

$$m_{s1} = \{m_{s1} \in m_{s0} | d_i.allMF.featureTypes = d_n.allMF.featureTypes\} \quad (5.13)$$

$$m_{s2} = \{m_{s2} \in m_{s1} | ratios(d_i.allMF.featureTypes) \simeq ratios(d_n.allMF.featureTypes)\} \quad (5.14)$$

$$m_{s3} = \{m_{s3} \in m_{s2} | d_i.singleMF \simeq d_n.singleMF\} \quad (5.15)$$

The filter for m_{s1} removes solutions whose associated data sets d_i do not have the feature types of the new data set d_n . For a new data set with numerical data, this filter removes ML solutions not trained on numerical data at all. Conversely, ML solutions trained with at least one numerical feature have similarity level 1. The filter for m_{s2} keeps only those solutions whose associated data sets d_i have feature type ratios similar to the feature type ratios of the new data set d_n . Ratios are considered similar if they are within one decile of the feature type ratios of the new data set. This means, e. g., removing ML solutions not trained on data sets with 95% +/- 5%-points of numerical features. The filter for m_{s3} compares the metafeatures in *singleMF*. Again, the solutions in m_{s3} must have *singleMF* values within one decile of the *singleMF* values of the new data set. This means that we, e. g., compare the percentage of outliers of numeric features in the new data set d_n to the percentage of outliers of the numeric features in associated data sets d_i in M . If the percentage of outliers of all numeric features is within +/- 5%, e. g., [0.2, 0.0, 0.11] and [0.18, 0.05, 0.07], the solution's similarity is level 3.

Step 1 passes over the ML solutions trained on the most similar data sets (m_s in Algorithm 5.1), i. e., having the highest similarity level. If m_s have a similarity lower than 3, the function *checkSimilarity()* adds *distrust points* and warnings to w_t . Formally, $w_t = \{distrustPoints, warnings(t)\}$, i. e., a scalar value *distrustPoints* and a list of short explanations *warnings(t)* describing a problematic condition t . w_t is the basis of a distrust score that estimates the suitability of the recommendations for the use case. It is inspired by the ML test score of Breck et al. (Breck et al., 2017). Distrust points are added for each problematic condition t that occurs at any step in AssistML. During this step, $w_t.distrustPoints$ can receive up to 3 distrust points depending on the highest similarity level found. Similarity level 0 awards 3 distrust points; similarity level 1 awards 2 distrust points and similarity level 2 awards 1 distrust point. No distrust points are awarded if the highest similarity level 3 is achieved.

5.4.2. Step 2: Identify Acceptable/Nearly Acceptable ML Solutions

The second step divides ML solutions from the previous step based on their performance into two groups, i. e., m_{ACC} and m_{NACC} . An ML solution belongs to m_{ACC} if its performance values for each relevant metric fall in the percentile range established by the user in $p_n.ranges$. For instance, a range of 0.2 for accuracy means that only the ML solutions of m_s which are among the upper 20% regarding accuracy are acceptable. For example, if the accuracy values of solutions in m_s vary uniformly between 0.7 and 0.92, ML solutions with an accuracy of 0.87 or higher are considered acceptable.

An ML solution belongs to m_{NACC} if its performance values are lower than those of solutions in m_{ACC} . Nevertheless, the performance values of m_{NACC} solutions may only be lower than the threshold for m_{ACC} solutions by the amount defined by $p_n.ranges$. In the same example with a 0.2 accuracy range, ML solutions in m_s belong to m_{NACC} if their accuracy values are lower than 0.87 and greater or equal than 0.82. ML solutions in m_{NACC} serve as comparison for ML solutions in m_{ACC} .

Algorithm 5.2 Identify [nearly] acceptable ML solutions

Require: $m_s, p_n.ranges$ **Ensure:** m_{ACC}, m_{NACC}, w_t

```
1: function CLUSTER( $m_s, ranges$ )
2:    $cls \leftarrow$  DBSCAN( $m_s$ )
3:    $mts \leftarrow m_s.p.metrics$ 
4:    $qlim \leftarrow p_n.ranges$ 
5:   for  $j = 1$  to  $[cls]$  do
6:      $accl = \max(mts(i) \in cls(j)) - qlim$ 
7:      $accFit(i) = \frac{|\{m_s | m_s \in cls(j) \wedge mts(i) \geq accl\}|}{|\{m_s | m_s \in cls(i)\}|}$ 
8:      $naccl = \max(mts(i) \in cls(j)) - 2 * qlim$ 
9:      $naccFit(i) = \frac{|\{m_s | m_s \in cls(j) \wedge mts(i) < accl \wedge mts(i) \geq naccl\}|}{|\{m_s | m_s \in cls(i)\}|}$ 
10:    if ( $accFit(i) \geq 51\%$ ) then
11:       $m_{ACC} \leftarrow cls(i)$ 
12:    else if ( $naccFit(i) \geq 51\%$ ) then
13:       $m_{NACC} \leftarrow cls(i)$ 
14:    end if
15:  end for
16:  if ( $m_{ACC} = \{\}$ ) then
17:    terminate ASSISTML
18:  else
19:     $w_t \leftarrow$  TESTFITS( $m_{ACC}, m_{NACC}$ )
20:    return  $m_{ACC}, m_{NACC}, w_t$ 
21:  end if
22: end function
```

The first task of [Algorithm 5.2](#) (line 2) uses the DBSCAN algorithm (Ester et al., 1996) to cluster the solutions in m_s based on their performance values, which are contained in $m_s.p.metrics$. The need for clustering derives from the fact that companies typically store hundreds of ML solutions in their repositories (R_1). Furthermore, ML solutions with different configurations can still produce very similar performance across several metrics. For instance, [Figure 5.3](#) illustrates 228 ML solutions developed for different use cases as colored dots in a three dimensional space. The color indicates the cluster to which the ML solution belongs, as determined by a clustering algorithm. The plot shows that ML solutions from different use cases and with

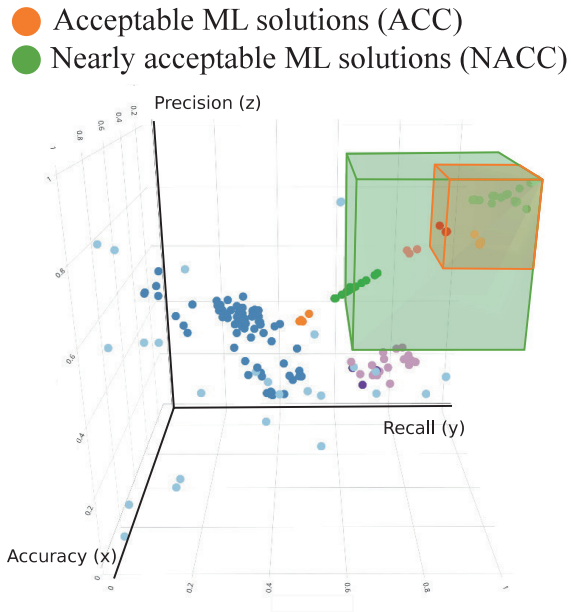


Figure 5.3.: Visualization of ACC and NACC groups where each dot represents an ML solution.

different configurations can have similar performance trade-offs, resulting in clearly separated clusters. The identification of **Acceptable (ACC)** and **Nearly Acceptable (NACC)** solutions is thereby facilitated. The consequence of these effects is that, after a relatively small number of ML solutions, it becomes inefficient to check and compare the performance of every ML solution. Clustering analysis reduces the number of necessary checks by grouping similarly performing ML solutions.

The DBSCAN algorithm is appropriate for this task for two reasons. First, it removes the need for citizen data scientists to predefine the number of clusters, thus removing a potential source of bias. Second, DBSCAN's minimum distance parameter guarantees that all ML solutions in a cluster have the same performance trade-off across all considered performance metrics.

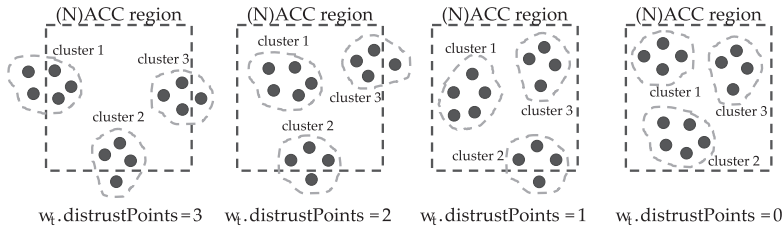


Figure 5.4.: Points assignment for the distrust score after clustering.

The second task (Lines 7 and 9) computes the fit of each cluster i in the ACC and in the NACC group. The fit of a cluster is the proportion of its ML solutions that are inside the acceptable (*accFit*) or nearly acceptable (*naccFit*) ranges. The third task (Lines 10 and 14) uses *accFit* and *naccFit* to assign the current cluster of ML solutions to either m_{ACC} or m_{NACC} . A value of 51% is used in both assignments to indicate a simple majority in the cluster.

Once all clusters have been processed, AssistML is interrupted if no cluster of ML solutions can be assigned to m_{ACC} . This means that no ML solution in m_s meets the performance preferences of the new use case. In such a situation, it is better to develop a new ML solution from scratch with senior ML experts. Otherwise, function *testFits()* adds distrust points and warnings to w_t depending on how well clusters fit in the groups m_{ACC} or m_{NACC} . The criteria to award distrust points are illustrated in Figure 5.4. If for each cluster, at least one ML solution is outside its m_{ACC} or m_{NACC} group, 3 distrust points are awarded. If the simple majority of clusters have at least one solution outside of a group, 2 distrust points are awarded. If this only holds for the simple minority of clusters, 1 distrust point is awarded. No distrust points are given if all ML solutions of all clusters are fully inside their group. A corresponding explanation is added to $w_t.warnings$ to describe the situation in each group.

Algorithm 5.3 Find ML solution patterns

Require: m_{ACC}, m_{NACC} **Ensure:** $rules_m$

```
1: function PATTERNS( $m_{ACC}, m_{NACC}$ )
2:    $settings \leftarrow m_{ACC}.s, m_{NACC}.s$ 
3:    $metricsLabels \leftarrow m_{ACC}.p.metricsLabels, m_{NACC}.p.metricsLabels$ 
4:    $rules_m \leftarrow \text{FPGROWTH}(settings, metricsLabels, minConf,$ 
      $minSupport)$ 
5:    $rules_m \leftarrow \text{REMOVEDUPLICATES}(rules_m)$ 
6:    $rules_m \leftarrow \{rules_m : confidence(rules_m) < 1 \ \&$ 
      $leverage(rules_m) > 0 \ \& \ lift(rules_m) > 1\}$ 
7:   return  $rules_m$ 
8: end function
```

5.4.3. Step 3: Find ML Solution Patterns

The third step searches for patterns in the metadata of the *ACC* and *NACC* solutions. For this purpose, [Algorithm 5.3](#) builds association rules with the settings and performance metadata of m_{ACC} and m_{NACC} solutions. Association rules describe patterns of the kind: "IF training time label is *D* (antecedent), THEN number of custom parameters is 5-10, has neural networks algorithm (consequent)". They indicate the common occurrence, not causality, of the antecedent and the consequent among ML solutions.

The first task of this step generates frequent item sets and rules using the FP-Growth algorithm (Han et al., 2000) (line 4 in [Algorithm 5.3](#)). Its input is metadata describing the configuration *settings* and the performance labels *metricsLabels* of both the acceptable and nearly acceptable ML solutions. The step combines metadata from both m_{ACC} and m_{NACC} to capture patterns supported by solutions in m_{ACC} and m_{NACC} , since these patterns can be decisive for the solution's performance. The step configures the FP-Growth algorithm by means of two parameters to prompt the creation of as many association rules as possible. The minimum confidence, which measures a rule's reliability, is set to 0.7 and minimum support to 0.25, which sets the threshold to consider a rule frequent. The result is a list $rules_m$, where each rule has confidence, leverage and lift values. Depending on the ML solutions

Table 5.3.: Sample ML solution recommendation report

1	RFR_kick_011	
2	Overall score: 0.9914	
3	Performance Labels	
4	Accuracy: A	Precision: A
5	Recall: A	Training Time: C
6	Output Explanation	
7	Feature-3 is suitable for the task. Feature-6 is unsuitable for the task.	
8	Data Preprocessing	
9	Categorical data is read via One-Hot Encoding (...)	
10	ML Solution Patterns	
11	IF ML solution has [random forests algorithm] and [number of custom parameters is 5 - 15] then [recall label is A] and [library used is sklearn].	
12	Deployment Description	
13	Deployed on single_host with 2 cores with 2.6 GHZ	
14	Language	python v. 3.6.0
15	Implementation	sklearn.ensemble. Extra-TreeClassifier v. 0.22.2
16	Nr. Dependencies: 6	Nr. Parameters: 14

contained in the metadata repository, it may be necessary to readjust these values.

The following task (line 5) removes duplicate rules with the same items and the same values of confidence, leverage and lift. This is because the two rules express the same co-occurrence pattern. The next task (line 6) removes low-quality rules from $rules_m$. Here, the step removes rules with a confidence value of 1 or with leverage of 0 or with lift of 1. These metric values indicate that rules are trivial or that the antecedent and consequent are statistically independent.

5.4.4. Step 4: Generate List of Recommendations

The final step generates a report containing a list of k ML solutions for both m_{ACC} and m_{NACC} . The positive and negative examples indicate a citizen data scientist the pros, cons, and effects of using different ML solution configurations. As a result, s/he can make informed decisions for the new use case. The lists are stratified by type of learning algorithm to ensure variety in the recommendations. The report additionally includes the original query q_n and w_t . *warnings* with explanations of problematic situations collected during the previous steps. Furthermore, it contains the $distrustScore_n$ to estimate how applicable a recommendation is. This distrust score is computed based on w_t . *distrustPoints* as shown in Equation (5.16)

$$distrustScore_n = \frac{\sum w_t \cdot distrustPoints}{t} \quad (5.16)$$

A dedicated recommendation report describes each selected ML solution in $m_{ACC}(k)$ and $m_{NACC}(k)$ individually. Table 5.3 shows a simplified version of this report for a sample recommendation. The report describes the ML solution's performance and configuration in a simple and intuitive way for citizen data scientists. An overall score summarizes the different performance values in *p.metrics*. For example, the overall score can be computed on accuracy, precision, recall, and training time. This score metric is the average of these four normalized metrics, scaled to a range from 0 (worst) to 1 (best). Also, the performance labels in *p.metricsLabels* further describe the solution's performance for each considered metric.

A global explanation of the ML solution's output based on *p.margins* completes the description of the solution's performance. This global explanation exemplifies the ML solution's behavior regarding certain data features. Data features with low margin values, e. g., lesser than 0.05, are considered unsuitable for the ML solution's task. Data features with high margin values, e. g., greater than 0.3, are considered suitable.

The report also describes the solution's configuration with metadata from m_{ACC} and m_{NACC} . First, the report includes the preprocessing used on each feature type. Any pattern from $rules_m$ that contains an element of the solution's configuration, e. g., the same implementation library, is added to the *ML Solution Patterns* section of the report. Data about the software and hardware resources needed to deploy the ML solution, the used programming language and algorithm implementation library, as well as the number of custom hyperparameters complete the report.

The generated reports are ranked using the overall score value. If there are ties, the individual performance metrics rank further involved ML solutions. A sequence deduced from the performance ranges in p_n of user query q_n determine the order in which to use the performance metrics for this ranking. Metrics with narrower ranges are assumed more important than metrics with wider ranges.

5.5. Prototype and Evaluation

The first part in this section describes the AssistML prototype. The second part explains the approach to evaluate the functionality of AssistML on the basis of two use cases. The third part discusses four types of evaluation results obtained with the aforementioned prototype and evaluation approach. The last part in the section assesses how AssistML fulfills the requirements from [Section 5.1.2](#). The source code and a short demonstration video of the AssistML prototype is available on GitHub ¹.

5.5.1. Prototypical Implementation

AssistML was implemented as a prototype consisting of modules developed in R and Python. Each module groups functionality corresponding to a step in AssistML. Together, they process a user query q_n and deliver in return a recommendation report $report_n$. [Figure 5.5](#) depicts the module architecture of the prototype. Arrows indicate the data and logic flow between the

¹<https://github.com/al-villanueva/assistml>

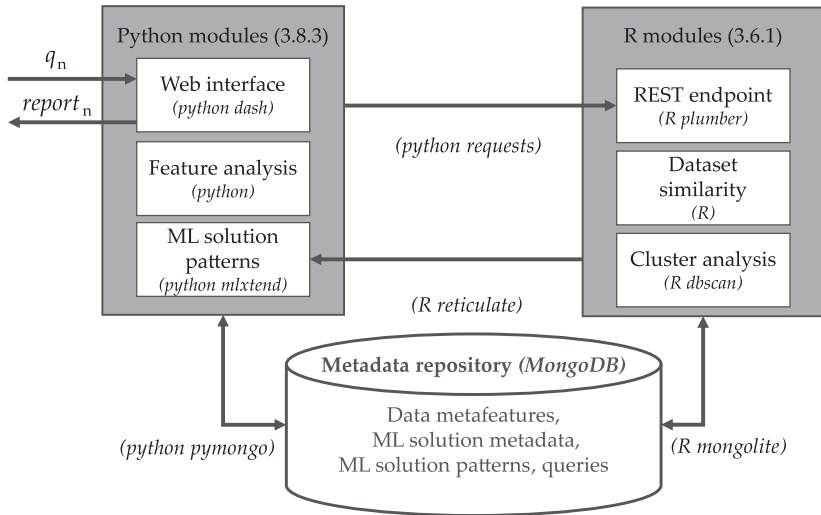


Figure 5.5.: Module architecture of the Assist ML prototype. Used libraries are shown in parentheses

modules. The required metadata repository was implemented as document collections in MongoDB. This allows the modules to use JSON documents as data exchange format. In order to ensure experimental reproducibility of the results, this dissertation describes the system requirements and necessary installation steps to run the AssistML prototype in [Appendix B](#).

Once installed, users can issue queries to the prototype via a web interface or via a REST API. [Figure 5.6a](#) shows the prototype’s web interface. Users introduce the query data using the controls on the sidebar ([Figure 5.6b](#)). After the user uploads a sample of the use case data, the *feature analysis* module computes the necessary metafeatures to compare the new data with the data sets documented in the repository. Query results are presented on the main section of the web interface. These are composed of a summary section, a performance visualization of all recommended ML solutions, and a list of recommendation reports (see [Figure 5.6a](#)).

Query results

Use case

Select a use case from the list or type a new one

Dataset Characteristics

Upload dataset as a csv file here

Drag and Drop or Select Files

steel-plates-fault.csv is uploaded successfully

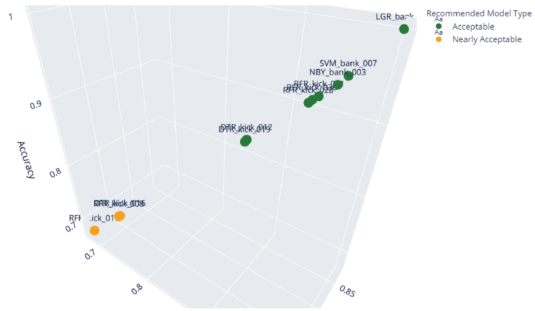
Select Label of Target Class

Select Datatype of Target Class

There is 8 acceptable models that match your query and 3 nearly acceptable models.

The distrust score for is: 55.559999999999995% and the reason for this is the following

- Dataset similarity level 1: Datasets used shared data types. Distrust Pts increased by 2
- The selection of NACC solutions was not as clean as possible. Distrust Pts increased by 3



(a) Web interface

Use case

Select a use case from the list or type a new one

Classifier Preferences

Output Type

Dataset Characteristics

Upload dataset as a csv file here

Drag and Drop or Select Files

steel-plates-fault.csv is uploaded successfully

Select Label of Target Class

Select Datatype of Target Class

Enter Feature Annotation List

Select Accuracy

Selected Accuracy: "0.1"

Select Precision

Selected Precision: "0.1"

Select Recall

Selected Recall: "0.1"

Select Training Time

Selected Training Time: "0.1"

ANALYSE DATASET

(b) Detail of the input controls

Figure 5.6.: Prototypical implementation of AssistML

Moreover, the evaluation approach included the development of 228 ML solutions to populate the metadata repository. These solutions represent a diverse selection of configurations in order to reflect the typical application scenario in companies. The solutions were developed for different data sets with various feature types. Their predictive models were trained using various learning algorithms from different libraries and programming languages. Table 5.4 gives specific details about the ML solutions in the metadata repository. The description of use case data includes their feature types in parentheses. If multiple feature types are included, they are listed from most common to least common in the data set. Several performance metrics were collected for each ML solution using 5-fold cross-validation. These include accuracy, precision, recall, training time, execution time for a single prediction, the f1 score and the confusion matrix.

Table 5.4.: Metadata repository contents

Element (count)	Description
Use cases (5)	kick auction (categorical,numeric) bank marketing (categorical,numeric) human activity recognition (numeric) gasdrift (numeric) amazon fine food reviews (numeric, unstructured text, categorical, date time)
ML tasks (2)	Binary classification, multi-class classification
ML algorithms (9)	Decision trees, random forests, neural networks, logistic regression, naive bayes, K nearest neighbors, gradient boosting machines, support vector machines, general linear model
Languages (2)	Python, R
Libraries (4)	scikit-learn, RWeka, pycaret, H2O

5.5.2. Evaluation Approach

The prototypical implementation enables the evaluation of the AssistML concept with a multistep approach. The following paragraphs explain the details of the use cases, the evaluation settings and the evaluation steps.

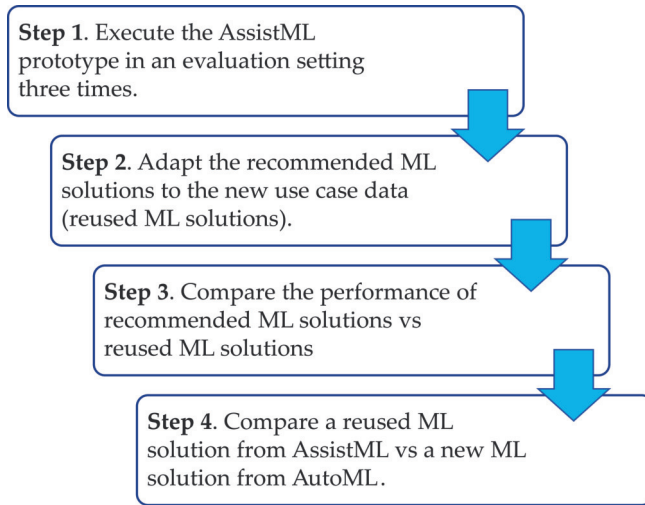


Figure 5.7.: Steps of the evaluation approach

The evaluation approach considers two new use cases, which are unknown to the metadata repository. The first and main use case deals with a fault detection task during *steel plates* production. It is based on a public data set for binary classification¹. As basis for comparison, the evaluation approach discusses a complementary use case based on the public data set *adult*². It describes a predictive task to determine if an adult has a yearly income over 50 000 dollars.

Figure 5.7 gives an overview of the four approach steps. The evaluation approach begins by issuing queries for each use case using two evaluation settings (see step 1 in Figure 5.7). These settings represent two levels of performance preferences. For simplicity, the same preferences are set across all performance metrics considered, i. e., accuracy, precision, recall, and training time. Settings *q-steel-10* and *q-adult-10* demand ML solutions to have the top 10% best values in all four metrics to be considered acceptable.

¹<https://www.openml.org/d/1504>

²<https://www.openml.org/d/1590>

Settings *q-steel-20* and *q-adult-20* describe less restrictive demands, i. e., ML solutions with the top 20% performance values are considered acceptable. In practice, switching from the top 10% to top 20% setting can be the result of dealing with a performance trade-off. In that case, the citizen data scientist may look for ML solutions with high values on a critical performance metric at the expense of others. Each evaluation setting is executed three times following a randomized plan in order to track the execution times to generate recommendations per evaluation setting.

Step 2 of the evaluation approach adapts the list of n recommended ML solutions on the new use case data, i. e., *steel plates* or *adult*. Thereby, the source code and configuration settings of the recommended solution are reused, i. e., the sampling strategy, seed values, hyperparameters, algorithm implementation and software dependencies. Thus, they are called *reused ML solutions*. Step 3 collects metadata to compare the performance of recommended ML solutions to that of reused ML solutions .

The last step of the evaluation approach used an AutoML system to generate a new competing ML solution for both use cases. *H2O AutoML 3.32.1.6* was used with default configurations. The execution of H2O AutoML was triggered from an R script in R 3.6.3 on Windows 10. H2O AutoML trains and cross-validates ML models with different learning algorithms (LeDell and Poirier, 2020). These include: gradient boosting machines, general linear models, random forests, and neural networks. It also trains stacked ensembles with combinations of these algorithms. The trained models are then ranked based on a single performance metric, which varies depending on the type of learning task. The step compares a new ML model produced by AutoML against a reused ML solution obtained with AssistML. This allows the discussion of the advantages and disadvantages of both AutoML and AssistML.

This approach is designed to evaluate the following four aspects: (a) the information provided for each recommended ML solution, (b) the concept's ability to provide good recommendations, (c) the time it takes to obtain them, and (d) the advantages and disadvantages of using this approach over AutoML.

5.5.3. Evaluation Results

Overall, the evaluation approach delivers the following results: (a) ML solution recommendation reports for both use cases, (b) the performance values of *reused ML solutions*, i. e., recommended ML solutions adapted to the new use case data, (c) the execution times of the AssistML prototype for each of the four evaluation settings, and (d) the highest-ranked ML model trained with an AutoML system for both use cases. The following paragraphs discuss each of these results. For simplicity, the discussion of specific results focuses on representative examples. The corresponding paragraph indicates this at the beginning. However, the complete list of recommendations generated for all four evaluation settings is shown in [Table 5.8](#), at the end of this subsection. Columns 6 to 8 in the table describe the performance of recommended ML solutions. Columns 10 to 12 describe the performance of the corresponding reused ML solutions.

ML solution recommendation report. The report shown in [Table 5.3](#) in [Section 5.4.4](#) recommends the ML solution RFR_kick_011 in the q-steel-10 setting. The performance labels intuitively show the performance of this recommended ML solution. They give users a sense of how the solution compares to other solutions for the same use case. For instance, the labels in lines 4 and 5 show that the recommended ML solution achieves overall good prediction performance at the cost of a longer training time.

Recommendation reports also indicate the ML solution strengths and weaknesses w. r. t. data features. Citizen data scientists can compare the suitable or unsuitable data features of existing ML solutions (see line 7) to the data features in their new use case. As a result, they can select only those that resemble the suitable ones. Data preprocessing information is given in line 9. In this example, one-hot-encoding is applied on categorical data. The information in lines 7 and 9, along with the source code in the repository, reduce the effort and time needed to prepare the new use case data.

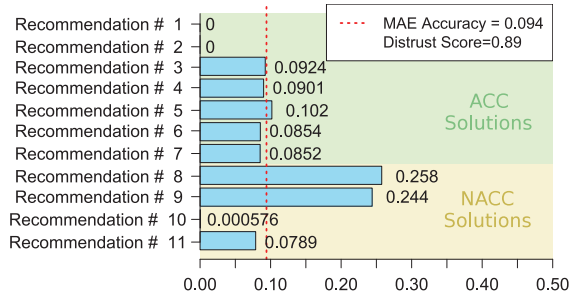
ML solution patterns offer global explanations (see [Section 5.2.3](#)). They provide the citizen data scientist with relevant relationships on configuration

and performance. For instance, the ML solution pattern in line 11 indicates that solutions with the Random Forest algorithm as well as 5 to 15 custom parameters tend to have a good recall (A label) and use the sklearn library. Citizen data scientists can restrict any adaptations they do on the ML solution to those that are indicated by these patterns. For instance, the pattern in line 11 helps them to avoid to tune more than the 15 parameters that are indicated by this pattern. This again reduces the time and effort they need for any adaptations. The report ends with deployment requirements (lines 13 to 16) to let the citizen data scientist decide if the solution can be deployed in the new use case.

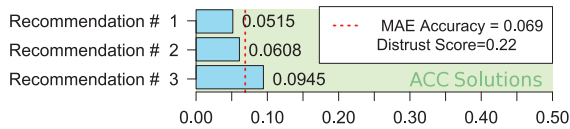
Performance comparison between recommended and implemented ML solutions. The discussion of this result focuses on evaluation settings q-steel-10 and q-adult-10. However, similar observations can be made for the other evaluation settings. The comparison in an evaluation setting includes the performance values of each recommended ML solution against the values of its corresponding reused ML solution. The recommended ML solution m_{rec} is one built for a previous use case, contained in the metadata repository, and described in a recommendation report, e. g., [Table 5.3](#). In this context, the performance of m_{rec} constitutes a prediction of using that combination of ML components on the new use case data. The reused ML solution m_{reu} is one that adapts the source code of the recommended ML solution to use it with the new use case data, in this case with the *steel plates* or *adult* data set. In other words, the performance of m_{reu} is the true value of using that combination of ML components on the new use case data.

Each pair of m_{rec} and m_{reu} in an evaluation setting is compared using the absolute error for the three performance metrics considered, namely accuracy absolute error (*AccAE*), precision absolute error (*PreAE*), and recall absolute error (*RecAE*). For instance, [Equation \(5.17\)](#) shows the formula for *AccAE*.

$$AccAE = |m_{rec}.acc_{rec} - m_{reu}.acc_{reu}| \tag{5.17}$$



(a) q-steel-10 setting.

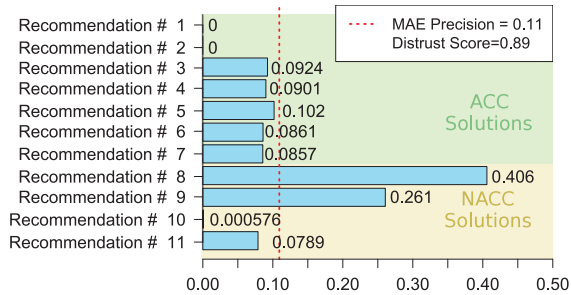


(b) q-adult-10 setting

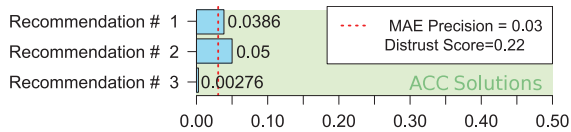
Figure 5.8.: Accuracy absolute error per recommendation ($AccAE$) in the q-steel-10 and q-adult-10 settings.

where $m.acc_{rec}$ and $m.acc_{reu}$ are the accuracy values of m_{rec} and m_{reu} , respectively. Similar equations can be defined for precision and recall. The absolute error quantifies the variation between the expected or predicted performance of an ML solution and the actual or observed performance. Therefore, smaller values of $AccAE$, $PreAE$ and $RecAE$ indicate better suitability of the recommendation for the new use case.

Figure 5.8a shows the values $AccAE$ for recommendations in the setting q-steel-10, from the highest ranked ACC recommendation (Recommendation # 1) to the lowest ranked NACC recommendation (Recommendation # 11). Figure 5.8b shows $AccAE$ for ACC recommendations in the q-adult-10 setting. In general, ACC recommendations lead to smaller, more consistent absolute errors than NACC recommendations. This confirms the suitability of ACC recommendations. They can be contrasted with NACC recommendations to analyze the small differences that contribute to their good performance. Moreover, the mean absolute error of accuracy (MAE Accuracy), shown as a dotted red line, summarizes the comparison for the whole evaluation



(a) q-steel-10 setting.



(b) q-adult-10 setting

Figure 5.9.: Precision absolute error per recommendation ($PreAE$) in the q-steel-10 and q-adult-10 settings.

setting. Notice that MAE Accuracy is bigger in setting q-steel-10, where the data set similarity is lower (level 1). AssistML warns the user about this with a high distrust score of 0,89. In setting q-adult-10, where the data set similarity is higher (level 2), the MAE and the distrust score provided by AssistML decrease, too. This behavior of the values confirms the ability of the distrust score to inform users about the error to expect when adapting the recommended ML solutions.

The values of $PreAE$ (see Figure 5.9) and $RecAE$ (see Figure 5.10) behave similarly in both use cases. This indicates that AssistML is able to establish similarity adequately, because the reused ML solutions in the new use case perform as the recommended solutions did in the original use cases across all performance metrics considered. Recommendation #8 for the evaluation setting q-steel-10 represents an exceptional case, because the absolute error reaches between 25 and 40 % in each performance metric. As shown in column 2 of Table 5.8, the corresponding NACC recommenda-

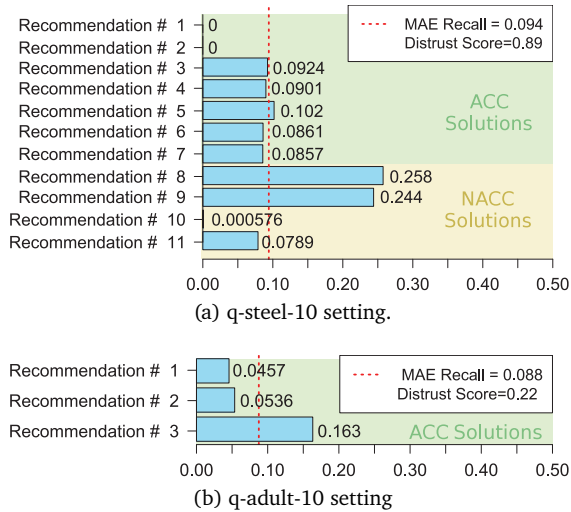


Figure 5.10.: Recall absolute error per recommendation ($RecAE$) in the q-steel-10 and q-adult-10 settings.

tion, *SVM_kick_004*, employs the same learning algorithm as the second ACC recommendation *SVM_bank_007*, i. e., support vector machines. However, they have different configurations and training settings, e. g., different implementation libraries, data sampling ratios and library dependencies. This highlights the importance of having both positive and negative recommendations to avoid generalizations about specific learning algorithms. NACC recommendations serve as counter examples with which citizen data scientists can better understand why ACC recommendations perform well.

Also note that the MAE in both evaluation settings is relatively small, i. e., between 3 and 11% across all performance metrics. This is even with the consideration of exceptional NACC recommendations. This is acceptable for citizen data scientists. It constitutes a negligible price to pay, since they can save huge efforts by avoiding the time-consuming implementation of completely new ML solutions. One alternative for citizen data scientists is to tune the hyperparameters and other settings to improve the performance

Table 5.5.: Execution times of the AssistML prototype. Values in seconds (s)

Evaluation setting	Mean (s)	Median (s)	Standard Dev. (s)
q-steel-10	30,906	26,822	8,376
q-steel-20	12,292	12,225	0,762
q-adult-10	41,102	41,058	1,292
q-adult-20	41,545	41,410	0,792

in the new use case. However, this additional effort may only lead to slight performance increases of about 1 to 2%-points for any of the metrics. Such an elaborate parameter tuning is usually not a practical option for citizen data scientists in light of such small performance increases. Another alternative is to use an AutoML system to obtain an optimized ML model without any user involvement. This however may imply several sacrifices, e. g., the simplicity of the model or the explanations about the model’s behavior. This alternative is further discussed at the end of this subsection.

Execution times to generate recommendations. The discussion of execution times focuses on the four evaluation settings. The results obtained confirm further the advantages of saving efforts and time during the development of ML solutions. The prototype delivers recommendation lists for both use cases in a matter of seconds. Table 5.5 shows the average and median execution times collected after carrying out a 3-fold randomized execution plan for all four evaluation settings. Due to variations in individual hardware used to run the prototype, execution times may vary to a certain extent. The execution time for any evaluation setting never surpasses 45 seconds, with standard deviation remaining also low. In some cases, it is even below 15 seconds.

These measurements concern the execution of single experiments. In a production environment, the demands for this assistant service may fluctuate, with demand peaks and bottoms. The prototypical implementation is thus as a stateless web service that can be scaled and replicated as necessary.

Criterion	AutoML	AssistML
Execution time	41 minutes	30,9 seconds + adaption time + 0,3 to 0,7 seconds training time
Number of models /recommendations	60	11
Performance (accuracy)	50% to 79,68%	85,33% to 100%
Highest ranked model/ solution	Stacked ensemble with 5 models: 1 deep learning, 2 random forests, 1 gradient boosting machines, 1 gen- eral linear model	Gaussian naive bayes classifier
Available information	6 performance metrics and confusion matrix	ML solution recommendation report

Table 5.6.: AutoML vs AssistML for the evaluation setting *q-steel-10*

Comparison to an ML model developed with AutoML. The discussion of this result focuses on evaluation settings *q-steel-10* and *q-adult-10*. AutoML represents an alternative for citizen data scientists to develop ML models. H2O’s AutoML (LeDell and Poirier, 2020) is applied on both the *steel plates* and *adult* data sets. H2O’s AutoML uses default settings to reflect the interest of the citizen data scientist to minimize input. AutoML results are compared to the reused ML solutions from the most demanding evaluation settings of AssistML for each use case, i. e., *q-steel-10* and *q-adult-10*. The comparison focuses on the advantages and disadvantages of using each development approach for the citizen data scientist. The comparison includes the execution time, the number of models available, the performance values of the models, their complexity, and the amount of information each approach provides.

Table 5.6 summarizes the output of both approaches for the *steel plates* use case. From it, it is clear that AutoML takes considerably longer than AssistML to deliver results. AutoML takes 41 minutes to deliver a trained ML model, whereas AssistML takes only 30,9 seconds to provide recommendations. However, AssistML still needs to adapt the source code of the recommended ML solution to the new use case data. This *adaptation time* varies depending on the user’s skills and the complexity of the solution. Thus, the adaptation time is indicated, but not quantified. Yet, AssistML provides information to facilitate this adaptation via the recommendation report, as

well as by providing access to the original source code in the repository. Afterwards, the ML model in the ML solution still needs to be trained. For the recommendations in evaluation setting q-steel-10, this takes negligible time.

Besides the highest ranked ML model, AutoML also gives a *leaderboard* of all the models it trained. This is done with the intention of offering options to the citizen data scientists. Yet, the number of additional models may become too big to analyze and compare comprehensively. AssistML takes this into consideration and caps the length of the list of recommendations to a sample, to avoid offering too many options.

Regarding the observed performance, the evaluation approach uses accuracy to compare models in AutoML's leaderboard to the reused solutions based on ACC recommendations from AssistML. The highest accuracy value of AutoML models is lower to the lowest accuracy value in AssistML. Moreover, the value range for AutoML models (29,68%-points) is wider than that of the AssistML ACC reused solutions (14,67%-points).

Regarding the complexity of the resulting model, the evaluation approach compares the highest ranked AutoML model to the first recommendation from AssistML. Here again there is a contrast between AutoML's more complex stacked ensemble and the simpler naive bayes classifier.

Finally, the citizen data scientist obtains different amounts of detail from each approach. AutoML provides six different performance metrics to describe each ML model in the leaderboard, as well as their confusion matrices. The performance metrics include area under the curve (AUC), mean square error (MSE), root-mean-square error (RMSE), Logloss, area under the precision and recall curve (AUCPR) and error/accuracy. The confusion matrix for each leaderboard's model can also be individually retrieved. In contrast, AssistML describes the ML solution regarding the data it uses, configuration and performance with the more intuitive recommendation report.

The results in this evaluation setting illustrate the disadvantages of using AutoML. For the sake of avoiding user intervention, AutoML incurs in much longer execution times and delivers many less-performing and more-

Criterion	AutoML	AssistML
Execution time	41,76 minutes	41.1 seconds + adaption time + 0,45 to 13,5 seconds training time
Number of models /recommendations	56	3
Performance (accuracy)	80,03% to 83,60%	75,64% to 84,90%
Highest ranked model/ solution	Stacked ensemble with 54 models: 26 deep learning, 2 random forests, 25 gradient boosting machines, 1 general linear model	Random forest classifier
Available information	6 performance metrics and confusion matrix	ML solution recommendation report

Table 5.7.: AutoML vs AssistML for the evaluation setting *q-adult-10*

complicated ML models. Additionally, these models are also difficult to understand for citizen data scientists only on the basis of expert ML metrics.

Table 5.7 summarizes the output of both approaches for the *adult* use case. In this evaluation setting, the execution time, number of models or recommendations and available information display similar behaviors as in the comparison for the *steel plates* use case.

Regarding the observed performance, the comparison shows that both approaches produce similar accuracy values with their best model or recommendation. It is important to note that the value range of AutoML models is narrower (3,57%-points) than that of AssistML ACC recommendations (9,26%-points). However, this is achieved at the expense of more complicated models. Specifically, AutoML’s highest ranked ML model is an ensemble stack of 54 individual models, whereas AssistML recommends a random forest classifier. Thus, for this evaluation setting, AutoML delivers a complicated ML model after 41 minutes which is 1,3%-points more accurate than the best recommendation of AssistML, the latter being found in only 41 seconds.

The results in this evaluation setting illustrate the importance of having suitable ML solutions to exploit the advantages of AssistML. A diverse number of ML solutions increases the probability of citizen data scientists to achieve acceptable performance intuitively and quickly. At the cost of adapting existing source code, they obtain a simpler ML solution, along with explanations after a few seconds.

Table 5.8.: Experimental data

1.Evaluation setting	2.Recomm. code	3.Recomm. group	4.Distrust score	5.Data set similarity	6.Accuracy	7.Precision	8.Recall	9.Adaptation code	10.Accuracy	11.Precision	12.Recall	13.Accuracy absolute error	14.Precision absolute error	15.Recall absolute error	
q-steel-10	NBY_bank_003	ACC	0,89	1	1,000	1,000	1,000	NBY_steelplates_001	1,000	1,000	1,000	0,000	0,000	0,000	
q-steel-10	SVM_bank_007	ACC	0,89	1	1,000	1,000	1,000	SVM_steelplates_001	1,000	1,000	1,000	0,000	0,000	0,000	
q-steel-10	RFR_kick_011	ACC	0,89	1	0,990	0,990	0,990	RFR_steelplates_001	0,897	0,897	0,897	0,092	0,092	0,092	
q-steel-10	RFR_kick_028	ACC	0,89	1	0,984	0,984	0,984	RFR_steelplates_003	0,894	0,894	0,894	0,090	0,090	0,090	
q-steel-10	RFR_kick_030	ACC	0,89	1	0,985	0,985	0,985	RFR_steelplates_002	0,883	0,883	0,883	0,102	0,102	0,102	
q-steel-10	DTR_kick_019	ACC	0,89	1	0,939	0,939	0,939	DTR_steelplates_002	0,853	0,853	0,853	0,085	0,086	0,086	
q-steel-10	DTR_kick_018	ACC	0,89	1	0,938	0,939	0,939	DTR_steelplates_003	0,853	0,853	0,853	0,085	0,086	0,086	
q-steel-10	SVM_kick_004	NACC	0,89	1	0,900	0,829	0,900	SVM_steelplates_003	0,642	0,423	0,642	0,258	0,406	0,258	
q-steel-10	NBY_kick_001	NACC	0,89	1	0,886	0,841	0,886	NBY_steelplates_003	0,642	0,580	0,642	0,244	0,261	0,244	
q-steel-10	DTR_kick_017	NACC	0,89	1	0,850	0,850	0,850	DTR_steelplates_004	0,850	0,850	0,850	0,001	0,001	0,001	
q-steel-10	RFR_kick_010	NACC	0,89	1	0,830	0,830	0,830	RFR_steelplates_004	0,909	0,909	0,909	0,079	0,079	0,079	
												Mean:	0,0942	0,1093	0,0943
q-steel-20	NBY_bank_003	ACC	0,56	1	1,000	1,000	1,000	NBY_steelplates_001	1,000	1,000	1,000	0,000	0,000	0,000	
q-steel-20	SVM_bank_007	ACC	0,56	1	1,000	1,000	1,000	SVM_steelplates_001	1,000	1,000	1,000	0,000	0,000	0,000	
q-steel-20	RFR_kick_011	ACC	0,56	1	0,990	0,990	0,990	RFR_steelplates_001	0,897	0,897	0,897	0,092	0,092	0,092	
q-steel-20	RFR_kick_028	ACC	0,56	1	0,984	0,984	0,984	RFR_steelplates_003	0,894	0,894	0,894	0,090	0,090	0,090	
q-steel-20	RFR_kick_030	ACC	0,56	1	0,985	0,985	0,985	RFR_steelplates_002	0,883	0,883	0,883	0,102	0,102	0,102	
q-steel-20	DTR_kick_019	ACC	0,56	1	0,939	0,939	0,939	DTR_steelplates_002	0,853	0,853	0,853	0,085	0,086	0,086	
q-steel-20	DTR_kick_018	ACC	0,56	1	0,938	0,939	0,939	DTR_steelplates_003	0,853	0,853	0,853	0,085	0,086	0,086	
q-steel-20	DTR_kick_016	NACC	0,56	1	0,757	0,757	0,757	DTR_steelplates_005	0,838	0,838	0,838	0,080	0,080	0,080	
q-steel-20	RFR_kick_008	NACC	0,56	1	0,751	0,751	0,751	RFR_steelplates_006	0,896	0,896	0,896	0,144	0,144	0,144	
q-steel-20	RFR_kick_019	NACC	0,56	1	0,675	0,675	0,675	RFR_steelplates_005	0,890	0,890	0,890	0,215	0,215	0,215	
												Mean:	0,0894	0,0895	0,0895
q-adult-10	RFR_bank_001	ACC	0,22	2	0,901	0,920	0,972	RFR_adult_002	0,849	0,881	0,926	0,052	0,039	0,046	
q-adult-10	DTR_bank_004	ACC	0,22	2	0,874	0,931	0,926	DTR_adult_002	0,813	0,881	0,872	0,061	0,050	0,054	
q-adult-10	NBY_bank_001	ACC	0,22	2	0,851	0,919	0,912	NBY_adult_004	0,756	0,916	0,748	0,094	0,003	0,163	
												Mean:	0,0689	0,0305	0,0876
q-adult-20	RFR_bank_001	ACC	0,11	2	0,901	0,920	0,972	RFR_adult_002	0,849	0,881	0,926	0,052	0,039	0,046	
q-adult-20	DTR_bank_004	ACC	0,11	2	0,874	0,931	0,926	DTR_adult_002	0,813	0,881	0,872	0,061	0,050	0,054	
q-adult-20	NBY_bank_001	ACC	0,11	2	0,851	0,919	0,912	NBY_adult_004	0,756	0,916	0,748	0,094	0,003	0,163	
												Mean:	0,0689	0,0305	0,0876

5.5.4. Assessment

This section concludes by determining whether any characteristic of AssistML fulfills the four practical requirements that citizen data scientists face to develop ML solutions.

Regarding **[R₁]**, the concept is expected to reuse information from existing ML solutions. AssistML fulfills this requirement with the use of a metadata repository containing previously developed ML solutions. The repository contains source code, training and test data to reproduce the results reported for each ML solution. Moreover, the metadata – summarized in the recommendation report – describe the necessary adaptations to the recommended ML solutions in detail. These resources significantly speed up the development of new solutions.

The concept fulfills **[R₂]**, as it provides recommendation reports (see [Table 5.3](#)) that explain the ML solutions globally, i. e., its overall behavior and composition. The reports describe the solution's performance and configuration in an intuitive manner. They thereby avoid the use of expert metrics and enable the comparison of good and bad configurations via comprehensible performance labels. In addition, they provide patterns to explain the ML solutions' functioning w. r. t. certain data features and data preprocessing techniques. In this regard, they offer citizen data scientists more intuitive explanations than the competing approach AutoML to understand and select ML solutions.

Regarding **[R₃]**, the concept is expected to provide recommendations in a responsive manner. AssistML has very short execution times when generating recommendations and thus fulfills this requirement. In addition, citizen data scientists only need to specify a minimum amount of information to issue queries q_n . These two factors allow citizen data scientists to assess the feasibility of multiple ML solutions efficiently. They can iterate on the recommendation process quickly and thereby, e. g., change their performance preferences. In this respect, AssistML offers citizen data scientists more flexibility than AutoML to explore different implementation alternatives.

AssistML fulfills **[R₄]**, as citizen data scientists may state preferences for multiple user-defined performance criteria in a query q_n . AssistML then considers all criteria simultaneously and offers intuitive performance labels for each of them. These labels clearly indicate the trade-offs an ML solution implies across different criteria. Citizen data scientists may use this to intuitively compare the recommended ML solution relative to others.

5.6. Summary and Future Work

This chapter introduced AssistML, a concept to recommend ML solutions for predictive use cases. This concept is subject to four practical requirements which are based on the needs of practitioners with less ML knowledge, e. g., citizen data scientists. AssistML analyzes metadata of existing ML solutions stored in a repository to find suitable matches for the user query. Furthermore, it offers intuitive explanations of the recommended ML solutions, e. g., by identifying frequent patterns among different data features. The chapter also presented a prototypical implementation of the concept as well as its evaluation with two use cases. The AssistML prototype is responsive, as it provides recommendations within 12 to at most 42 seconds. In addition, the finally implemented ML solutions show a performance that is very close to the performance of the recommendations, with a MAE of at most 10% across all performance metrics. Moreover, citizen data scientists can adapt and reuse the recommendations with reduced effort and time thanks to the source code and metadata available in the repository. Compared to AutoML, AssistML offers citizen data scientists simpler, intuitively-explained ML solutions in considerably less time. Moreover, these solutions perform as well as or even better than AutoML models.

Overall, AssistML provides very promising results, which can serve as basis for a new development approach for ML solutions in organizations wishing to use ML in their use cases. Possible future work directions include the analysis of the queries issued to AssistML to guide the expansion of the metadata repository. For instance, frequently used performance criteria

can be turned into preference patterns that show the need for a particular type of ML solution. An example of this can be criteria favoring solutions with high accuracy and low recall can indicate the interest in more complex ensemble methods. Another direction is the automated adaptation of a recommendation into the new use case. This can be achieved by enforcing a naming scheme based on feature types. This can allow data features from very similar data sets to be used with the existing ML solution without any need for further time-consuming adaptations.

CHAPTER
6

CONCLUSION AND FUTURE WORK

This chapter draws general conclusions about all contributions presented in this dissertation. The first section summarizes the research contributions and assesses the extent to which they fulfill the research challenges (see [Section 1.3](#)). The second section discusses future research directions that build upon the research contributions.

6.1. Assessment of the Research Contributions

This dissertation proposes methods and concepts to solve the ineffectiveness problem of current ML solution development processes. Based on a division of the ML solution development process into three steps, this dissertation proposes four research challenges that need to be addressed to solve the research problem just mentioned. The four challenges are: [CH-1](#), i.e., the design challenge, which requires concepts and methods to systematize the collaboration of different roles to specify ML solutions from end-to-end;

CH-2, i. e., the configuration challenge, which requires concepts and methods to ensure that all information needed to evaluate, compare, reproduce and reuse ML solutions is systematically collected; CH-3, i. e., the selection challenge, which requires concepts and methods to facilitate the comparison, understanding and selection of ML solutions by ML non-experts that nonetheless are concerned by use case preferences, i. e., decision makers and citizen data scientists; and CH-4, i. e., the process challenge, which requires concepts and methods to ensure that all necessary development activities, i. e., the activities that fulfill the interests of all roles throughout the three main steps, are performed in the right sequence. This section discusses the fulfillment of these four research challenges by means of the research contributions presented in Chapters 3 to 5. Table 6.1 gives an overview of this assessment.

The adapted design methodology AD4ML addresses the design challenge CH-1. AD4ML formalizes ML solution specifications as systems of design equations, with relationships between design elements expressed as non-zero coefficients in each design equation. These mathematical formalism allows specifications made with AD4ML to be processed and analyzed by software systems. For instance, the prototypical *ML solution designer* can identify which components are missing the contributions of certain roles, e. g., a data scientist to specify an analytics concept. It can also issue warnings whenever a matching or decomposition operation violates one of the methodology's axioms. AD4ML allocates to each role in the development team a design element to contribute to the specification, i. e., DRs for domain experts, ACs for data scientists and TRs for software developers. Constraints, although not a design element, allow decision makers to set boundary conditions that affect the overall development project, e. g., by setting total development costs. Thereby, AD4ML clearly defines the level of involvement that is expected from each role and ensures that an ML solution designed with AD4ML has a clear trace between the domain-specific requirements that justify the use of ML to the specific software libraries and values that are used to implement it. This makes it easy to identify the impact that every software component of the ML solution has on the use case.

The metadata profiles and the solution viewer of the ML solution profiling framework address the configuration challenge [CH-2](#). The metadata profiles, i. e., the profile for the use case task (ATP), the profile for data characteristics (DQP), the profile for the hardware and software resources (AIP), and the profile for the ML model configuration (ACP), serve as thematic metadata sets to capture all information needed to reproduce the ML solution's data, ML model, and hardware and software stack. Moreover, the metadata profiles are implemented as schema-free key-value documents, which nevertheless allows the use of identification codes and naming conventions. Software developers can thus refer to specific metadata profiles using only their identification codes to find out which components are part of an ML solution, how were they implemented and the performance they delivered. This introduces an abstraction level when referring to ML solution components, which allows data scientists and software developers to work with more complex specifications easily.

The recommendation process AssistML addresses the selection challenge [CH-3](#). The automated analysis that is performed to identify the ACC and NACC ML solutions automates the consideration of multiple performance criteria and trade-offs. AssistML can explore the data set similarity and performance suitability of multiple ML solutions and identify any useful performance pattern to recommend citizen data scientists and decision makers which ML solutions can be refactored for reuse in a new use case. For this purpose, AssistML provides recommendation reports with intuitive descriptions. They allow citizen data scientists to ponder the advantages and disadvantages of using an ML solution, as well as to compare many alternatives that can be built with different software libraries, ML algorithms, data sets, tasks and/or programming languages. Overall, AssistML allows decision makers and citizen data scientists to make better informed selections in less time.

The development process of the ML solution framework along with the metadata repository from AssistML address the process challenge [CH-4](#). The sequence of activities indicates each role in the development team when to get involved, the information and components available at that

stage and the deliverables they have to provide for the development to continue. The process also contemplates activities to interact with the other contributions, e. g., the ML solution designer from AD4ML or the AssistML solution repository. For instance, the process indicates when to secure an identification code for a new ML solution and when to document each component in the solution repository. This turns the documentation of metadata into a milestone activity that secures the configuration of that component for other roles to work with. The process also brings a technical standard to track the progress in the development project, as the activities it describes are required for different types of ML solutions.

In sum, it can be concluded that the research contributions presented throughout this dissertation do address the research challenges that make the development of ML solutions ineffective. This has been confirmed through the implementation and evaluation of each contribution with exemplary use cases.




Contribution	Design challenge (CH-1)	Configuration challenge (CH-2)	Selection challenge (CH-3)	Process challenge (CH-4)
	Agile design methodology. Formal solution specifications. Methods to validate, assess and visualize specifications. ML solution designer.			
		Metadata profiles to ensure reproducibility and reusability. Systematic identification of solution components. ML solution viewer facilitates overview of complex solutions.		ML solution development process coordinates work of team roles and sequences development steps.
			Recommendation process facilitates the reuse of ML solutions. Recommendation reports provide intuitive information to citizen data scientists.	ML solution repository standardizes collection of ML solution components.

Table 6.1.: Fulfillment of the challenges by the main contributions of this dissertation

6.2. Future Research Directions

The work on the research contributions discussed in the previous section has led to identify new interesting areas where research can continue. These research directions are detailed below.

6.2.1. ML Solution Assessment Function

The suitability of ML solutions depends on their ability to deliver their predictive capability with a certain expected performance that domain experts require for the given use case. However, ML solutions consist of many components, e. g., data preprocessors, prediction models, or sample data sets. Therefore, the performance observed when evaluating the ML solution depends on their combined performance and in last instance on the individual (hyper)parameters used to configure them. Currently, to test if a component combination and parameterization leads to acceptable performance, that combination must be implemented and evaluated. Any new proposed combination, or a slight modification of existing ones, requires both development and evaluation to be carried out again. This implies a significant amount of development effort. At the moment, AssistML provides the performance of existing ML solutions in previous use cases as an indication of the performance that can be expected in the new use case. However, these indications necessarily include an error component. To address this inefficient situation, there is the need for a concept or method to predict the performance of an ML Solution before it is developed.

The ML solution framework can comprehensively document the configuration and performance of ML solution components. Metadata generated with this framework enable the development of a Bayesian Network to predict the performance or performance ranges that a combination of ML solution components will produce. The architecture of the network can be determined by analyzing the correlation of each configuration element in the ML solution, e. g., the hyperparameters, the preprocessing steps, the type of ML algorithm used, or the type of data features used. Depending on the number

of significant correlations found, one or multiple networks can be modeled. The conditional probabilities to train the network can be computed once a sufficient number of different ML solutions are documented in a metadata repository. In this context, the consideration of performance ranges can help increase the number of available examples, specially considering the low likelihood that two ML solutions for completely different use cases and/or learning on different data sets produce the exact same accuracy, precision or recall.

Once tested, the assessment function can be applied on an ML solution specification at the end of the design step. In this situation, the assessment function could serve as an early filter to decide the viability of continuing ML solution development. An early and accurate detection of underperforming ML solutions can greatly decrease development times and, as a consequence, free up time and resources to cover more ML solution development projects.

LIST OF AUTHOR PUBLICATIONS

Publications as first author

Villanueva Zacarias, A. G., Weber, C., Reimann, P., Mitschang, B. (2021). AssistML: A concept to recommend ML solutions for predictive use cases. 2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA), 2021, pp. 1-12, doi: 10.1109/DSAA53316.2021.9564168.

Villanueva Zacarias, A. G., Ghabri, R., Reimann, P. (2021). Leveraging Axiomatic Design to Improve the Design Process of Machine Learning Solutions in Manufacturing. Accepted for publication in a special issue of the International Journal of Semantic Computing (IJSC). To appear.

Villanueva Zacarias, A. G., Ghabri, R. and Reimann, P. (2020). AD4ML: Axiomatic Design to Specify Machine Learning Solutions for Manufacturing. 2020 IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI) (p./pp. 148-155), August, 2020. DOI: 10.1109/IRI49571.2020.00029

Villanueva Zacarias, A. G., Reimann, P. and Mitschang, B. (2018). A framework to guide the selection and configuration of machine-learning-

based data analytics solutions in manufacturing. *Procedia CIRP*, 72, 153–158. DOI: 10.1016/j.procir.2018.03.215

Villanueva Zacarias, A. G., Kassner, L. and Mitschang, B. (2017). Exploring Text Classification Configurations - A Bottom-up Approach to Customize Text Classifiers based on the Visualization of Performance. *Proceedings of the 19th International Conference on Enterprise Information Systems*, : SCITEPRESS - Science and Technology Publications. DOI: 10.5220/0006309705040511

Supervised student theses and projects

Master Theses. ACP Dashboard: An interactive visualization tool for selecting Analytics Configurations in an industrial setting. June - December 2017

Relevance of the two adjusting screws in data analytics: data quality and optimization of algorithms. January - July 2017

Student Projects. Research Project INFOTECH. Use and Evaluation of Bayesian Networks to Predict the Performance of ML Solutions. Winter Semester 2020/2021.

Study Project INFOTECH. Performance Analysis of Machine Learning Models (MLM-Perf). Winter Semester 2019/2020.

Practical Course Information Systems. The Machine Learning Algorithm Wars (ML-Wars). Summer Semester 2019.

Developed prototypes

ML Solution Viewer. A dashboard to visualize the metadata of developed ML solutions. Introduced in [Chapter 3](#).

ML Solution Designer. A web client and back end to specify ML solutions using the AD4ML methodology. Introduced in [Chapter 4](#).

ML Solution Repository. A MongoDB database, consisting of metadata collections, as well as source code, and training data samples to consolidate the ML solutions developed by different projects. Introduced in [Chapter 5](#).

AssistML. A recommendation system with a web interface and API interface to recommend ML solutions to citizen data scientists. Introduced in [Chapter 5](#).

BIBLIOGRAPHY

- Adler, P., C. Falk, S. A. Friedler, T. Nix, G. Rybeck, C. Scheidegger, B. Smith, S. Venkatasubramanian (2018). ‘Auditing black-box models for indirect influence’. In: *Knowledge and Information Systems* 54.1, pp. 95–122 (Cited on p. 121, 122).
- Akao, Y., B. King (1990). *Quality function deployment: integrating customer requirements into product design*. Vol. 21. Productivity press Cambridge, MA (Cited on p. 47, 63).
- Alpaydin, E. (2009). *Introduction to machine learning*. 2nd. MIT Press (Cited on p. 45, 46).
- Anderl, R., M. Eigner, U. Sendler, R. Stark (2012). *Smart engineering: interdisziplinäre Produktentstehung*. Springer-Verlag (Cited on p. 37–39).
- Ashmore, R., R. Calinescu, C. Paterson (2021). ‘Assuring the machine learning lifecycle: Desiderata, methods, and challenges’. In: *ACM Computing Surveys (CSUR)* 54.5, pp. 1–39 (Cited on p. 21, 22).
- Atwal, H. (2020). ‘The Problem with Data Science’. In: *Practical DataOps: Delivering Agile Data Science at Scale*. Berkeley, CA: Apress, pp. 3–26. URL: https://doi.org/10.1007/978-1-4842-5104-1_1 (Cited on p. 17).
- Azevedo, A. I. R. L., M. F. Santos (2008). ‘KDD, SEMMA and CRISP-DM: a parallel overview’. In: *IADS-DM* (Cited on p. 49).
- Baier, L., F. Jöhren, S. Seebacher (May 2019). ‘Challenges in the deployment and operation of machine learning in practice’. In: *Proceedings of the 27th European Conference on Information Systems (ECIS2019)* (Cited on p. 29, 117, 118).
- Bank, M., R. Remus, M. Schierle, P. S. Ag, N. Calzolari, K. Choukri, T. Declerck, M. U. Doğan, B. Maegaard, J. Mariani, A. Moreno, J. Odiijk, S. Piperidis (2012). ‘Textual Characteristics for Language Engineering.’ In: *Proceedings of the Eight In-*

- ternational Conference on Language Resources and Evaluation (LREC'12). European Language Resources Association (ELRA), pp. 515–519 (Cited on p. 124).
- Bernardi, L., T. Mavridis, P. Estevez (2019). '150 Successful Machine Learning Models: 6 Lessons Learned at Booking.Com'. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '19. Anchorage, AK, USA: Association for Computing Machinery, pp. 1743–1751. URL: <https://doi.org/10.1145/3292500.3330744> (Cited on p. 21, 22, 26, 117, 118).
- Biondi, G. O., R. C. Prati (2015). 'Setting parameters for support vector machines using transfer learning'. In: *Journal of Intelligent & Robotic Systems* 80.1, pp. 295–311 (Cited on p. 120, 121).
- Bischi, B., P. Kerschke, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Fréchette, H. Hoos, F. Hutter, K. Leyton-Brown, K. Tierney, et al. (2016). 'Aslib: A benchmark library for algorithm selection'. In: *Artificial Intelligence* 237, pp. 41–58 (Cited on p. 55).
- Bishop, C. M. (2006). 'Pattern recognition'. In: *Machine learning* 128.9 (Cited on p. 46).
- Bourrasset, C., F. Boillod-Cerneux, L. Sauge, M. Deldossi, F. Wellenreiter, R. Bordawekar, S. Malaika, J.-A. Broyelle, M. West, B. Belgodere (2019). 'Requirements for an Enterprise AI Benchmark'. In: *Performance Evaluation and Benchmarking for the Era of Artificial Intelligence*. Ed. by R. Nambiar, M. Poess. Cham: Springer International Publishing, pp. 71–81 (Cited on p. 50).
- Braschler, M., T. Stadelmann, K. Stockinger (2019). *Applied data science: lessons learned for the data-driven business*. Springer International Publishing (Cited on p. 19).
- Breck, E., S. Cai, E. Nielsen, M. Salib, D. Sculley (2017). 'The ML test score: A rubric for ML production readiness and technical debt reduction'. In: *2017 IEEE International Conference on Big Data (Big Data)*, pp. 1123–1132 (Cited on p. 118, 130).
- Burkart, N., M. F. Huber (2020). *A Survey on the Explainability of Supervised Machine Learning*. arXiv: 2011.07876 [cs.LG] (Cited on p. 114, 117, 121, 122).
- Chapman, P., J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, R. Wirth, et al. (2000). 'CRISP-DM 1.0: Step-by-step data mining guide'. In: *SPSS inc 9*, p. 13 (Cited on p. 47, 107, 108).

- Choudhary, A. K., J. A. Harding, M. K. Tiwari (2009). 'Data mining in manufacturing: a review based on the kind of knowledge'. In: *Journal of Intelligent Manufacturing* 20.5, p. 501 (Cited on p. 16).
- Crowe, T. J., C.-C. Cheng (1996). 'Using quality function deployment in manufacturing strategic planning'. In: *International Journal of Operations & Production Management* 16.4, pp. 35–48 (Cited on p. 48).
- Demšar, J. (2006). 'Statistical comparisons of classifiers over multiple data sets'. In: *Journal of Machine learning research* 7.Jan, pp. 1–30 (Cited on p. 54).
- Dogan, N., Z. Tanrikulu (2013). 'A comparative analysis of classification algorithms in data mining for accuracy, speed and robustness'. In: *Information Technology and Management* 14.2, pp. 105–124 (Cited on p. 54).
- Eigner, M. (2013). 'Modellbasierte Virtuelle Produktentwicklung auf einer Plattform für System Lifecycle Management'. In: *Industrie 4.0: Beherrschung der industriellen Komplexität mit SysLM*. Ed. by U. Sendler. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 91–110. URL: https://doi.org/10.1007/978-3-642-36917-9_6 (Cited on p. 38, 42).
- Eigner, M., R. Stelzer (2009). *Product lifecycle management: Ein Leitfaden für product development und life cycle management*. Springer Science & Business Media (Cited on p. 37–39).
- Ester, M. et al. (1996). 'A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise'. In: *2nd International Conference on Knowledge Discovery and Data Mining*, pp. 226–231 (Cited on p. 131).
- Ethayarajh, K., D. Jurafsky (2020). 'Utility is in the Eye of the User: A Critique of NLP Leaderboards'. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 4846–4853 (Cited on p. 117, 118).
- Fayyad, U., G. Piatetsky-Shapiro, P. Smyth (1996). 'From data mining to knowledge discovery in databases'. In: *AI magazine* 17.3, p. 37 (Cited on p. 49).
- Feurer, M., K. Eggenberger, S. Falkner, M. Lindauer, F. Hutter (2020). 'Auto-sklearn 2.0: The next generation'. In: *arXiv preprint arXiv:2007.04074* (Cited on p. 118, 119).
- Feurer, M., A. Klein, K. Eggenberger, J. Springenberg, M. Blum, F. Hutter (2015). 'Efficient and robust automated machine learning'. In: *Advances in Neural Information Processing Systems*, pp. 2962–2970 (Cited on p. 54).
- Flaounas, I. (2017). 'Beyond the technical challenges for deploying Machine Learning solutions in a software company'. In: *Proceedings of the Human in the Loop Machine*

- Learning Work- shop, International Conference on Machine Learning*, (Cited on p. 18, 21, 22, 115, 117).
- Gandomi, A., M. Haider (2014). 'Beyond the hype: Big data concepts, methods, and analytics'. In: 35 (2), pp. 137–144 (Cited on p. 17, 43, 44).
- Giebler, C., C. Gröger, E. Hoos, H. Schwarz, B. Mitschang (2019). 'Leveraging the Data Lake: Current State and Challenges'. In: *Big Data Analytics and Knowledge Discovery*. Ed. by C. Ordonez, I.-Y. Song, G. Anderst-Kotsis, A. M. Tjoa, I. Khalil. Cham: Springer International Publishing, pp. 179–188 (Cited on p. 43).
- Gijbbers, P., E. LeDell, J. Thomas, S. Poirier, B. Bischl, J. Vanschoren (2019). 'An open source AutoML benchmark'. In: *6th ICML Workshop on Automated Machine Learning* (Cited on p. 119).
- Goldstein, A., A. Kapelner, J. Bleich, E. Pitkin (2015). 'Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation'. In: *Journal of Computational and Graphical Statistics* 24.1, pp. 44–65 (Cited on p. 121).
- Grabowski, H., R.-S. Lossack, J. Weißkopf (2002). *Datenmanagement in der Produktentwicklung: automatische Klassifikation von Produktdaten aus 3D-CAD-Systemen, PDM- und ERP-Systemen, XML- und Office-Dokumenten, ...* Deutsch. Literaturverz. S. 233 - 236. München ; Wien: Hanser, 248 S (Cited on p. 38–40).
- Gröger, C. (2018). 'Building an Industry 4.0 Analytics Platform'. In: *Datenbank-Spektrum* 18.1, pp. 5–14 (Cited on p. 18, 83, 113).
- Grover, P., A. K. Kar (2017). 'Big data analytics: A review on theoretical contributions and tools used in literature'. In: *Global Journal of Flexible Systems Management* 18.3, pp. 203–229 (Cited on p. 43, 44).
- Gupta, D. (2018). *Applied analytics through case studies using Sas and R: implementing predictive models and machine learning techniques*. Apress (Cited on p. 17).
- Haasis, S., D. Frank, B. Rommel, M. Weyrich (2003). 'Feature-based Integration of Product, Process and Resources'. In: *Feature Based Product Life-Cycle Modelling: IFIP TC5 / WG5.2 & WG5.3 Conference on Feature Modelling and Advanced Design-for-the-Life-Cycle Systems (FEATS 2001) June 12–14, 2001, Valenciennes, France*. Ed. by R. Soenen, G. J. Olling. Boston, MA: Springer US, pp. 93–108. URL: https://doi.org/10.1007/978-0-387-35637-2_6 (Cited on p. 38, 40, 41).
- Han, J., J. Pei, Y. Yin (May 2000). 'Mining Frequent Patterns without Candidate Generation'. In: *SIGMOD Rec.* 29.2, pp. 1–12 (Cited on p. 134).

- Henelius, A., K. Puolamäki, A. Ukkonen (2017). 'Interpreting classifiers through attribute interactions in datasets'. In: *Proceedings of the ICML Workshop on Human Interpretability in Machine Learning 2017 (WHI 2017)* (Cited on p. 121, 122).
- Herzwurm, G., U. Dowie, S. Schockert (Jan. 2001). 'Quality Planning for E-Commerce Applications'. In: *TAE-Konferenz* (Cited on p. 109).
- Herzwurm, G., W. Pietsch, S. Schockert, T. Tauterat (2012). 'QFD for Cloud Computing'. In: *Proceedings of the 18th International Symposium on Quality Function Deployment. International Symposium on Quality Function Deployment (ISQFD). Yamanashi, Japan* (Cited on p. 48).
- Herzwurm, G., S. Schockert, T. Tauterat (2015). 'Quality Function Deployment in Software Development-State-of-the-art'. In: *Proceedings of the 21th International Symposium on Quality Function Deployment* (Cited on p. 48, 107, 109).
- Hevner, a. R., S. T. March, J. Park (2004). 'Design Science in Information Systems Research'. In: *MIS Quarterly* 28 (1), pp. 75–105 (Cited on p. 31).
- Hoos, H. H. (2008). *Computer-aided design of high-performance algorithms*. Tech. rep. Technical Report TR-2008-16, University of British Columbia, Department of Computer Science (Cited on p. 54).
- Hoos, H. H., F. Neumann, H. Trautmann (2017). 'Automated Algorithm Selection and Configuration (Dagstuhl Seminar 16412)'. In: *Dagstuhl Reports*. Vol. 6. 10. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (Cited on p. 51, 55).
- Hutter, F., H. H. Hoos, K. Leyton-Brown, T. Stützle (2009). 'ParamILS: an automatic algorithm configuration framework'. In: *Journal of Artificial Intelligence Research* 36.1, pp. 267–306 (Cited on p. 55).
- Isermann, R. (2017). 'Supervision, fault-detection and fault-diagnosis methods – a short introduction'. In: *Combustion Engine Diagnosis: Model-based Condition Monitoring of Gasoline and Diesel Engines and their Components*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 25–47. URL: https://doi.org/10.1007/978-3-662-49467-7_2 (Cited on p. 75).
- Kashyap, P. (2018). *Machine Learning for Decision Makers: Cognitive Computing Fundamentals for Better Decision Making*. Apress (Cited on p. 45).
- Kotsiantis, S. B., I. Zaharakis, P. Pintelas, et al. (2007). 'Supervised machine learning: A review of classification techniques'. In: *Emerging artificial intelligence applications in computer engineering* 160.1, pp. 3–24 (Cited on p. 46).

- Kotsiantis, S. B., I. D. Zaharakis, P. E. Pintelas (2006). ‘Machine learning: a review of classification and combining techniques’. In: *Artificial Intelligence Review* 26.3, pp. 159–190 (Cited on p. 46, 53).
- Kubat, M. (2017). *An introduction to machine learning*. Springer (Cited on p. 46).
- Kühn, A., R. Joppen, F. Reinhart, D. Röltgen, S. von Enzberg, R. Dumitrescu (2018). ‘Analytics Canvas—A Framework for the Design and Specification of Data Analytics Projects’. In: *Procedia CIRP* 70. 28th CIRP Design Conference, pp. 162–167 (Cited on p. 48).
- Kuwajima, H., H. Yasuoka, T. Nakae (2020). ‘Engineering problems in machine learning systems’. In: *Machine Learning* 109.5, pp. 1103–1126 (Cited on p. 15).
- Langley, P., H. A. Simon (1995). ‘Applications of machine learning and rule induction’. In: *Communications of the ACM* 38.11, pp. 54–64 (Cited on p. 53, 70).
- LeDell, E., S. Poirier (July 2020). ‘H2O AutoML: Scalable Automatic Machine Learning’. In: *7th ICML Workshop on Automated Machine Learning (AutoML)*. URL: https://www.automl.org/wp-content/uploads/2020/07/AutoML_2020_paper_61.pdf (Cited on p. 142, 149).
- Li, C., A. Dakkak, J. Xiong, W.-M. Hwu (2019). ‘MLModelScope: Evaluate and Introspect Cognitive Pipelines’. In: *2019 IEEE World Congress on Services, SERVICES 2019, Milan, Italy, July 8-13, 2019*. Ed. by C. K. Chang, P. Chen, M. Goul, K. Oyama, S. Reiff-Marganiec, Y. Sun, S. Wang, Z. Wang. IEEE, pp. 335–338. URL: <https://doi.org/10.1109/SERVICES.2019.00093> (Cited on p. 49).
- Lieber, D., M. Stolpe, B. Konrad, J. Deuse, K. Morik (2013). ‘Quality prediction in interlinked manufacturing processes based on supervised & unsupervised machine learning’. In: *Procedia CIRP* 7, pp. 193–198 (Cited on p. 54).
- Mäkinen, S., H. Skogström, E. Laaksonen, T. Mikkonen (2021). ‘Who Needs MLOps: What Data Scientists Seek to Accomplish and How Can MLOps Help?’ In: *2021 IEEE/ACM 1st Workshop on AI Engineering - Software Engineering for AI (WAIN) of 43rd International Conference on Software Engineering (ICSE)* (Cited on p. 17, 43).
- Marchand, D. A., J. Peppard (2013). ‘Why IT fumbles analytics’. In: *Harvard Business Review* 91.1, pp. 104–112 (Cited on p. 76).
- Mattson, P., V. J. Reddi, C. Cheng, C. Coleman, G. Damos, D. Kanter, P. Micikevicius, D. Patterson, G. Schmuelling, H. Tang, G.-Y. Wei, C.-J. Wu (2020). ‘MLPerf: An Industry Standard Benchmark Suite for Machine Learning Performance’. In: *IEEE Micro* 40.2, pp. 8–16 (Cited on p. 50).

- Mell, P., T. Grance, et al. (2011). 'The NIST definition of cloud computing'. In: (Cited on p. 65).
- Mikkonen, T., J. K. Nurminen, M. Raatikainen, I. Fronza, N. Mäkitalo, T. Männistö (2021). 'Is Machine Learning Software Just Software: A Maintainability View'. In: *International Conference on Software Quality*. Springer, pp. 94–105 (Cited on p. 21).
- Mitchell, T. M. (1997). *Machine Learning*. Vol. 1. McGraw-Hill Series in Computer Science. Boston, Mass. [u.a.]: WCB/McGraw-Hill, XVII, 414 pages (Cited on p. 45).
- Monostori, L. (2003). 'AI and machine learning techniques for managing complexity, changes and uncertainties in manufacturing'. In: *Engineering applications of artificial intelligence* 16.4, pp. 277–291 (Cited on p. 20, 21).
- Moyne, J., J. Iskandar (2017). 'Big data analytics for smart manufacturing: Case studies in semiconductor manufacturing'. In: *Processes* 5.3, p. 39 (Cited on p. 16, 17, 43, 44).
- Nalchigar, S., E. Yu (Feb. 2020). 'Designing business analytics solutions'. In: *Business & Information Systems Engineering* 62.1.1, pp. 61–75. URL: <https://doi.org/10.1007/s12599-018-0555-z> (Cited on p. 48, 108, 110).
- O'Donovan, P., K. Leahy, K. Bruton, D. T. O'Sullivan (2015). 'An industrial big data pipeline for data-driven analytics maintenance applications in large-scale smart manufacturing facilities'. In: *Journal of Big Data* 2.1, p. 25 (Cited on p. 17, 43).
- Olson, R. S., R. J. Urbanowicz, P. C. Andrews, N. A. Lavender, J. H. Moore, et al. (2016). 'Automating biomedical data science through tree-based pipeline optimization'. In: *European Conference on the Applications of Evolutionary Computation*. Springer, pp. 123–137 (Cited on p. 118).
- Olson, R. S., W. La Cava, P. Orzechowski, R. J. Urbanowicz, J. H. Moore (Dec. 2017). 'PMLB: a large benchmark suite for machine learning evaluation and comparison'. In: *BioData Mining* 10.1, p. 36 (Cited on p. 54).
- Paley, A., R.-G. Urma, N. D. Lawrence (2020). *Challenges in Deploying Machine Learning: a Survey of Case Studies*. arXiv: 2011.09926 [cs.LG] (Cited on p. 22, 116, 117).
- Pan, S. J., Q. Yang (2010). 'A Survey on Transfer Learning'. In: *IEEE Transactions on Knowledge and Data Engineering* 22, pp. 1345–1359 (Cited on p. 119).
- Pham, D., A. Afify (2005). 'Machine-learning techniques and their applications in manufacturing'. In: *Proceedings of the Institution of Mechanical Engineers, Part B*:

- Journal of Engineering Manufacture* 219.5, pp. 395–412 (Cited on p. 16, 17, 21, 51, 53, 70).
- Raina, R., A. Y. Ng, D. Koller (2006). ‘Constructing informative priors using transfer learning’. In: *Proceedings of the 23rd international conference on Machine learning*, pp. 713–720 (Cited on p. 120).
- Ramos, A. L., J. V. Ferreira, J. Barceló (2011). ‘Model-based systems engineering: An emerging approach for modern systems’. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.1, pp. 101–111 (Cited on p. 42).
- Rauch, E., D. T. Matt, P. Dallasega (2016). ‘Application of axiomatic design in manufacturing system design: a literature review’. In: *Procedia CIRP* 53. 10th International Conference on Axiomatic Design, pp. 1–7 (Cited on p. 78).
- Saaty, R. (1987). ‘The analytic hierarchy process - what it is and how it is used’. In: *Mathematical Modelling* 9.3, pp. 161–176 (Cited on p. 64).
- Schneider, F., L. Balles, P. Hennig (2019). ‘DeepOBS: A Deep Learning Optimizer Benchmark Suite’. In: *International Conference on Learning Representations*. URL: <https://openreview.net/forum?id=rJg6ssC5Y7> (Cited on p. 50).
- Schwaber, K., J. Sutherland (2017). *The Scrum Guide™. The definitive guide to scrum: The rules of the game. November 2017*. URL: <https://www.scrumguides.org> (Cited on p. 95).
- Sculley, D., G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, D. Dennison (2015). ‘Hidden technical debt in machine learning systems’. In: *Advances in neural information processing systems* 28, pp. 2503–2511 (Cited on p. 15, 18, 20).
- Sebastiani, F. (2002). ‘Machine learning in automated text categorization’. In: *ACM Computing Surveys* 34 (1), pp. 1–47 (Cited on p. 44, 124).
- Sharp, M., R. Ak, T. Hedberg Jr (2018). ‘A survey of the advancing use and development of machine learning in smart manufacturing’. In: *Journal of Manufacturing Systems* (Cited on p. 20, 21, 47, 73).
- Shearer, C. (2000). ‘The CRISP-DM model: the new blueprint for data mining’. In: *Journal of data warehousing* 5.4, pp. 13–22 (Cited on p. 22, 47, 108).
- Snoek, J., H. Larochelle, R. P. Adams (2012). ‘Practical Bayesian optimization of machine learning algorithms’. In: *Advances in Neural Information Processing Systems*, pp. 2951–2959 (Cited on p. 55).

- Sokolova, M., G. Lalpalme (July 2009). 'A systematic analysis of performance measures for classification tasks'. In: *Information Processing & Management* 45.4, pp. 427–437 (Cited on p. 54, 123).
- Subianto, M., A. Siebes (2007). 'Understanding discrete classifiers with a case study in gene prediction'. In: *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. IEEE, pp. 661–666 (Cited on p. 121, 122).
- Suh, N. P., ed. (2001). *Axiomatic design: advances and applications*. Englisch. The MIT-Pappalardo series in mechanical engineering. New York ; Oxford: Oxford University Press, XXIII, 503 Seiten (Cited on p. 32, 74, 78–81, 84, 90, 92, 93).
- Tao, F., Q. Qi, A. Liu, A. Kusiak (2018). 'Data-driven smart manufacturing'. In: *Journal of Manufacturing Systems* 48. Special Issue on Smart Manufacturing, pp. 157–169. URL: <https://www.sciencedirect.com/science/article/pii/S0278612518300062> (Cited on p. 17, 42, 43).
- Van Rijn, J. N., F. Hutter (2018). 'Hyperparameter importance across datasets'. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2367–2376 (Cited on p. 120).
- Vanschoren, J. (2018). 'Meta-Learning: A Survey'. In: *arXiv preprint arXiv:1810.03548* (Cited on p. 114, 119, 123).
- Vanschoren, J., J. N. van Rijn, B. Bischl, L. Torgo (June 2014). 'OpenML: Networked Science in Machine Learning'. In: *SIGKDD Explor. Newsl.* 15.2, pp. 49–60. URL: <http://doi.acm.org/10.1145/2641190.2641198> (Cited on p. 49, 119).
- Viaene, S., A. Van den Bunder (2011). 'The secrets to managing business analytics projects'. In: *MIT Sloan Management Review* 53.1, p. 65 (Cited on p. 27, 76).
- Villanueva Zacarias, A. G., R. Ghabri, P. Reimann (2020). 'AD4ML: Axiomatic Design to Specify Machine Learning Solutions for Manufacturing'. In: *2020 IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI)*, pp. 148–155 (Cited on p. 30, 74).
- Villanueva Zacarias, A. G., R. Ghabri, P. Reimann (2021). *Leveraging Axiomatic Design to Improve the Design Process of Machine Learning Solutions in Manufacturing*. to appear (Cited on p. 30, 74).
- Villanueva Zacarias, A. G., P. Reimann, B. Mitschang (2018). 'A framework to guide the selection and configuration of machine-learning-based data analytics solutions in manufacturing'. In: *Procedia CIRP* 72. 51st CIRP Conference on Manufacturing Systems, pp. 153–158 (Cited on p. 31, 52).

- Villanueva Zacarias, A. G., C. Weber, P. Reimann, B. Mitschang (2021). ‘AssistML: A Concept to Recommend ML Solutions for Predictive Use Cases’. In: *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 1–12 (Cited on p. 31, 114).
- Villanueva Zacarias, A. G., L. Kassner, B. Mitschang. (2017). ‘Exploring Text Classification Configurations - A Bottom-up Approach to Customize Text Classifiers based on the Visualization of Performance’. In: *Proceedings of the 19th International Conference on Enterprise Information Systems - Volume 3: ICEIS, INSTICC*. SciTePress, pp. 504–511 (Cited on p. 31).
- Wagstaff, K. L. (June 2012). ‘Machine Learning that Matters’. In: *Proceedings of the 29th International Conference on Machine Learning*. Edinburgh, Scotland (Cited on p. 50, 117, 118).
- Walden, D. D., G. J. Roedler, K. Forsberg (2015). ‘INCOSE Systems Engineering Handbook Version 4: Updating the Reference for Practitioners’. In: *INCOSE International Symposium*. Vol. 25. 1. Wiley Online Library, pp. 678–686 (Cited on p. 42).
- Weber, C., P. Hirmer, P. Reimann, H. Schwarz (2019). ‘A New Process Model for the Comprehensive Management of Machine Learning Models’. In: *Proceedings of the 21st International Conference on Enterprise Information Systems - Volume 1: ICEIS*. Ed. by J. Filipe, M. Smialek, A. Brodsky, S. Hammoudi. SciTePress, pp. 415–422 (Cited on p. 21, 22).
- Westkämper, E., C. Löffler (2016). *Strategien der Produktion: Technologien, Konzepte und Wege in die Praxis*. Springer (Cited on p. 15, 16).
- Wuest, T., D. Weimer, C. Irgens, K.-D. Thoben (2016). ‘Machine learning in manufacturing: advantages, challenges, and applications’. In: *Production & Manufacturing Research* 4.1, pp. 23–45 (Cited on p. 15, 46, 51, 53, 70).
- Xin, D., E. Y. Wu, D. J.-L. Lee, N. Salehi, A. Parameswaran (May 2021). ‘Whither AutoML? Understanding the Role of Automation in Machine Learning Workflows’. In: *Proceedings of the Conference on Human Factors in Computing Systems (CHI ’21)*. 8-13. ACM. ACM (Cited on p. 18, 28, 29, 118, 119).
- Zaharia, M., A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, et al. (2018). ‘Accelerating the Machine Learning Lifecycle with MLflow.’ In: *IEEE Data Eng. Bull.* 41.4, pp. 39–45 (Cited on p. 18, 28, 48, 49, 115, 117, 119, 124).

All URLs were last checked on 30.06.2021.

LIST OF FIGURES

1.1. Elements of an ML solution. Core components have double line borders.	20
1.2. Typical ML solution development process.	21
1.3. Typical roles in ML solution development team.	23
1.4. Common elements in ML solution development.	25
1.5. Detail of the research challenges occurring during the development process of ML solutions.	27
1.6. Contributions to address the challenges in the development of ML solutions.	31
2.1. Overview of the product conception phase. After descriptions from (Anderl et al., 2012) and (Eigner and Stelzer, 2009) . .	38
2.2. Feature tree showing the connection between part feature, process and resources. After (Haasis et al., 2003)	41
2.3. General operation of Machine Learning	45
3.1. Motivation Scenario	52
3.2. <i>ML Solution Development Process</i> : A structured approach to create an ML solution, where experts are supported on particular topics	59

3.3. Metadata Profiles of the ML Solution Framework	64
3.4. System Architecture of the ML Solution Viewer	68
3.5. Web prototype of the ML Solution Viewer	69
3.6. ML solution viewer displaying details of performance metadata	70
4.1. Overview of the example use case for <i>Fault Detection</i>	75
4.2. Overview of the milestones (in black) and requirements (in gray) in the design process of Machine Learning Solutions	77
4.3. Conceptual overview of Axiomatic Design along with the roles involved in their definition.	79
4.4. Overview of AD4ML concepts.	83
4.5. List of initial Domain Requests for the use case example.	85
4.6. Example of the zigzagging processes to specify details of the initial and abstract DR-i-2. <i>Matching arrows</i> are dashed and black-headed. <i>Decomposition arrows</i> are dotted and white-headed. Decomposed design elements are shown as white boxes.	87
4.7. Flow diagram of the ML solution specification introduction in Section 4.3.2	92
4.8. Comparison between correctly and incorrectly designed ML solution specifications	97
4.9. Module architecture of the ML solution designer. The programming languages and libraries used are shown in parentheses.	102
4.10. ML solution designer	104
4.11. Detail of the analysis text available on the ML solution designer	105
5.1. Application scenario for recommendations of ML solutions.	116
5.2. Overview of the steps in AssistML	127
5.3. Visualization of ACC and NACC groups where each dot represents an ML solution.	132
5.4. Points assignment for the distrust score after clustering.	133

5.5. Module architecture of the Assist ML prototype. Used libraries are shown in parentheses	138
5.6. Prototypical implementation of AssistML	139
5.7. Steps of the evaluation approach	141
5.8. Accuracy absolute error per recommendation (<i>AccAE</i>) in the q-steel-10 and q-adult-10 settings.	145
5.9. Precision absolute error per recommendation (<i>PreAE</i>) in the q-steel-10 and q-adult-10 settings.	146
5.10. Recall absolute error per recommendation (<i>RecAE</i>) in the q-steel-10 and q-adult-10 settings.	147
A.1. Sample metadata of the ML solution "KNN_kick_001"	190

LIST OF ALGORITHMS

5.1. Select solutions on data similarity	128
5.2. Identify [nearly] acceptable ML solutions	131
5.3. Find ML solution patterns	134

LIST OF TABLES

4.1. Comparison of Axiomatic Design and AD4ML	82
4.2. Coverage of feasibility requirements by related approaches	109
5.1. Assessment of related approaches	120
5.2. Exemplary metafeatures by feature type	124
5.3. Sample ML solution recommendation report	135
5.4. Metadata repository contents	140
5.5. Execution times of the AssistML prototype	148
5.6. AutoML vs AssistML for the evaluation setting <i>q-steel-10</i>	149
5.7. AutoML vs AssistML for the evaluation setting <i>q-adult-10</i>	151
5.8. Experimental data	152
6.1. Fulfillment of the challenges by the main contributions of this dissertation	161
B.1. Software version numbers	192

ACRONYMS

ACP Analytics Configuration Profile. 61, 64, 65, 68

AIP Analytics Infrastructure Profile. 61, 64

ATP Analytics Task Profile. 61, 65

CAD Computer Aided Design. 40

CAE Computer Aided Engineering. 40

CAM Computer Aided Manufacturing. 40

CH-1 Design Challenge. 25, 30

CH-2 Configuration Challenge. 30

CH-3 Selection Challenge. 27, 31

CH-4 Process Challenge. 28, 32

CRISP-DM Cross Industry Standard Process for Data Mining. 45

CRM Customer Relationship Management. 40

DQP Data Quality Profile. 61, 63

ERP Enterprise Resource Planning. 40, 41

KDD Knowledge Discovery in Databases Process. 46, 47

MBSE Model Based Systems Engineering. 40

MES Manufacturing Execution System. 40, 41

PDM Product Data Management. 40

QFD Quality Function Deployment. 45, 62

SCM Supply Chain Management. 40

SEMMA Sample, Explore, Modify, Model, Assess. 47



SAMPLE METADATA PROFILES

The metadata profiles presented in [Section 3.3](#) were implemented as a single JSON document. This implementation facilitated the mass collection necessary to build the ML solution repository described in [Section 5.3](#).

[Figure A.1](#) visualizes an example JSON document containing the metadata profiles for the ML solution "KNN_kick_001". The document is generated via annotation commands in the solution source code. The document is organized in three main fields, i. e., "Data_Meta_Data" corresponding to the task description (*ATP*) and the data quality (*DQP*) profiles, "Training_Characteristics" corresponding to the configuration (*ACP*) and infrastructure (*AIP*) profiles, and "Metrics" complementing the configuration (*ACP*) profile. An additional field "Info" facilitates the identification of the JSON document in the solution repository.

The visualization was generated with PlantUML 1.2021.10¹

¹<https://plantuml.com/json>

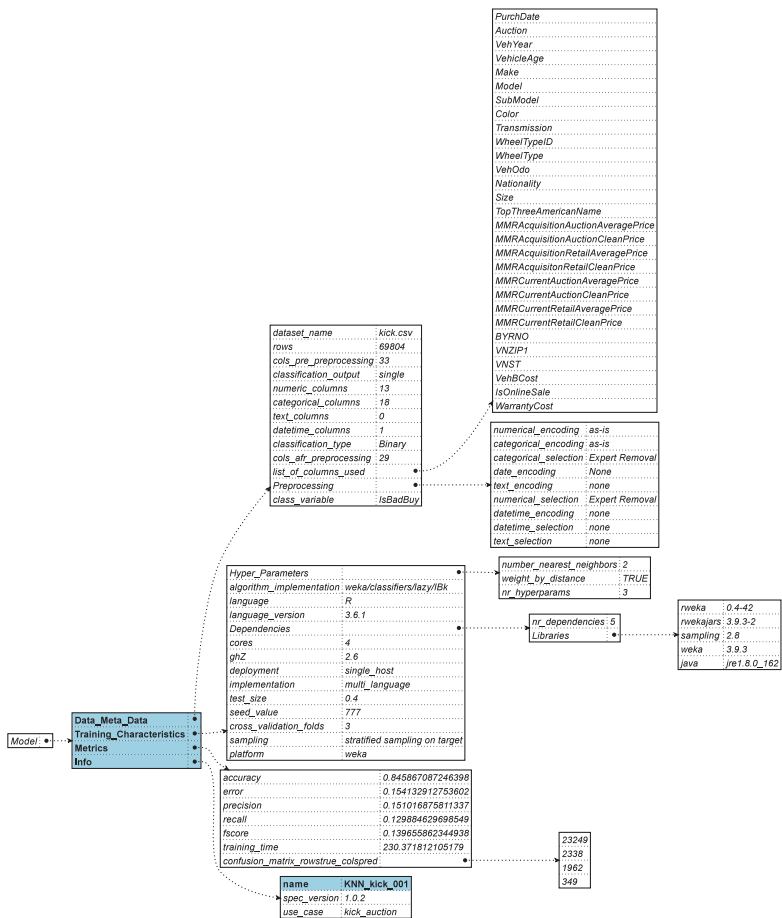


Figure A.1.: Sample metadata of the ML solution "KNN_kick_001"



SYSTEM REQUIREMENTS AND INSTALLATION GUIDE FOR ASSISTML

System Requirements. [Table B.1](#) shows the software and package dependencies required to set up the prototype. For our individual setup, we used an Ubuntu 18.04 host with 2 CPUs at 2.5 GHz, 8 GB of memory and 40 GB of disk space.

Installation guide. The following steps need to be carried out to run the prototype of AssistML and reproduce the experimental setup:

1. Clone GitHub repository.
2. Satisfy dependencies.
3. Create repository in Mongo with the provided data.
4. Launch the API ("assist.R") and web interface ("assist_dashboard.py").

Table B.1.: Software version numbers

Software	Version	URL
R	"3.6.3"	https://cran.r-project.org
plumber	"0.4.6"	https://www.rplumber.io
dbscan	"1.1-5"	https://cran.r-project.org/web/packages/dbscan
mongolite	"2.2.0"	https://cran.r-project.org/web/packages/mongolite
stringr	"1.4.0"	https://stringr.tidyverse.org
rjson	"0.2.20"	https://cran.r-project.org/web/packages/rjson
reticulate	"1.18"	https://rstudio.github.io/reticulate
Python	"3.8.6"	https://www.python.org/downloads/release/python-386
mlxtend	"0.17.3"	http://rasbt.github.io/mlxtend
requests	"2.24.0"	https://2.python-requests.org/en/master
pymongo	"3.11.0"	https://github.com/mongodb/mongo-python-driver
MongoDB	"4.4.0"	https://www.mongodb.com/try/download/community

5. In a web browser go to <http://localhost:8050>
6. To reproduce evaluation setting *q-steel-20* (see [Section 5.5.1](#)), watch the video *q-steel-20.mkv* in the repository. Other evaluation settings can be reproduced in a similar manner.